On the Nearly Spherical Stratified Flame Propagation

by

Charles Anthony Scudiere

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jyh-Yuan Chen, Chair
Professor Carlos Fernández-Pello
Professor Per-Olof Persson

Fall 2019

On the Nearly Spherical Stratified Flame Propagation

Abstract

On the Nearly Spherical Stratified Flame Propagation

by

Charles Anthony Scudiere

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor Jyh-Yuan Chen, Chair

Stable and reliable power is critical for not only modern conveniences, but also for basic goods and services needed to ensure protection of both life and property. To ensure a sustainable source of reliability in the global energy sector through current and future environmental and political changes and in concert with alternative and renewable production sources, current and future combustible fuels are needed to be accurately modeled. A key proponent of both natural gas and biogas is methane which has both current natural sources and future supply prospects. However there are still many fundamental questions regarding accurate modeling of the combustion of methane, and in particular within inhomogeneous mixtures. These stratification layers are less well understood in combustion environments than flames propagating through homogeneous mixtures, despite many of the current uses of this gaseous fuel in a variety of engineering systems.

A set of spherical methane-air experiments within a constant volume chamber using Schlieren imaging and pressure traces as well as supporting one dimensional and three dimensional numerical modeling was undertaken to explore stratified flames propagating through methane-air mixtures. With comparisons to past work, an investigation of the effects of the stratification layer's impact on the observed flame speed, product gas emissions, and to evaluate the possibility of extending the lean limit. To process the Schlieren images, a robust in-house edge tracking code was developed to track the progress of the flame observed in Schlieren images and closely evaluate the transient dynamics of the flame that occur within a flame burning through a stratification layer gradient set up between two mixture concentrations using a soap bubble. A speed up on the order of 20% higher than homogeneous equivalence ratio of 1.1 was observed in the rich to lean stratified cases. The experiments and numerical work agreed reasonably well with past experimental and numerical work. Higher CO was noted while burning in a stratified environment, while lower unburnt hydrocarbons and moderately lower NO_x was also noted from stratification layers compared with an equivalent homogeneous mixture. The lean limit appeared to be extended, and a discussion is given in light of the prior work and capabilities within this work.

While relative agreement was achieved experimentally with recent work, unexpected instabilities were noted in the flame that are difficult to be accounted for with the setup alone. This work adds to the possibility first mentioned and observed by Markstein, Behrens, and Einbinder of an inherent instability within stratified methane-air flames.

To my friends and family, immediate and extended.

Thank you for you discussions, patience, advice, support and understanding.

Contents

List of Figures

List of Tables

Acknowledgments

I cannot express enough gratitude to a variety of individuals and groups who have helped me through this journey both professionally and personally through out graduate school. Only with their support was this all possible.

First to my advisor, Professor J.Y. Chen, who has provided an array of support and collaboration through my work, and a wonderful research advisor. I am deeply grateful for this support and to have the freedom to pursue my interests, fail, improve, and follow wherever the research took me. Your patience, knowledge, and skill are unmatched.

To the various undergraduate and high school professors and teachers who have instilled a desire to continually learn, grow, and pursue physical understanding and knowledge for the greater good. Most especially Professor Paul Erickson, Professor Linton Corruccini, Randy Harris, Professor Mohamed Hafez, Professor Fidelis Eke, Professor Ian Kennedy, Professor Donald Margolis, and James Schaaf.

To the advice, friendship, collaboration, mentorship, and support I have received from lab mates within the Combustion Modeling Lab through out the years, including visiting scholars and students who I have shared both space and time with. Labmates Ben Wolk, Terry McCullum, Colin Hurlbut, Yumin Huang, Tiernan Casey, Yulin Chen, Daniel Pineda, Sirui Fu, Jorge Moreno, Xian Shi, Je Ir Ryu, Alex Frank, and Bhaskar Chaturvedi. Visiting students and scholars Jonathan Timo Lipkowicz, Fredrik Grøvdal, Farouk Chourou, Shenghao Yu, Niek van Rooij, Jakob Fochler, Catharina van Gool, Yasutaka Tomomatsu, Shuguo Shi, and Bowen Zheng. Visiting scholars Dr. Tao Chen, Dr. Bo Xu, and Dr. Xiao Gan.

I would also like to express a heartfelt appreciation for Professor Carlos Fernández-Pello and the members and visitors of the Combustion Fire Processes laboratory who have mentored, supported, and/or guided me in various ways through my progress. Lab members Casey Zak, Daniel Murphy, Sarah Scott, Shmuel Link, James Urban, Mari Thomsen, Lauren Gagnon, and Andy Rodriguez among other past lab members. Including visiting students and scholars Xinyan Huang, Yudong Liu, Yong Lu, Simon Santamaria, and Jeanette Cobian.

I would also like to thank a number of volunteering undergraduate students who have provided their assistance through out different portions of this work, designing, modeling, experimenting, or building. Including Nathan Lebherz, Quoc Ho, Jonathan Anglingdarma, and Jordan Tate. Your time, eagerness to learn, and energy have been an invaluable gift.

The technical staff of which the machining for and of the experimental setup would be significantly more challenging and time consuming, and so I must acknowledge the assistance from Michael Neufer, Scott McCormick, Alex Jordan, Jacob Gallego, Jeff Higginbotham, Dennis Lee, Pete Graham, Mick Franssen, Jesse Lopez, Tom Clark, and Daniel Paragas. I would also like to thank the ME administrative and student services staff including Yawo Dagbevi Akpawu, Shareena Samson, Irma Paz-Viray, Rene Viray, Isabel Blanco, and Victoria Garcia.

I would also like to thank a number of funding bodies who, with their support, made it possible for me to continue financially. Both through the CEC and the NSF, along with all of the efforts and extended support by my advisor Professor J.Y. Chen and Professor Robert

Dibble, Tim Sennot, Miguel Aznar, and MaryAnne Peters. Additionally, support through the graduate division with block grants as well as through the ME department and Statistics department with GSI funding opportunities.

There are also numerous other members of the greater UC Berkeley community who have provided friendship and collaboration through my PhD including Ravi Singh, Jim Oreluk, Bradley Cage, Amin Ghafari, Sarah Frank, Shanna Hays, Saghi Sadoughi, Claire Wemp, Jessica Avva, Brian Mick, Jan Mehnen, Patrick Hylton, Tholfaqar Al Emara, Saahil Agrawal, Jessica Lee, and Kay Dawson among numerous wonderful people. Your enduring guidance, advocacy, friendship, discussions, support and advise through this journey has made all the difference.

Outside Berkeley, there have been a number of individuals who have helped in various ways through my graduate school career. First, I would like to acknowledge my family who have in their own way entered on this journey with me, most especially my parents Amelia and Charles. Your enduring support and opportunities you have provided laid the foundation I have relied on through out graduate school. And to my sister who has always been there to share and work through various challenges along the way.

There are also a number of significant friendships that have helped, encouraged, and supported me through various parts of graduate school including the decision, application, and through out various significant graduate school milestones. Including Charlene Lumanlan, Becca Sagastegui, Julia Taussig, Claudia López, Katherine Aronson-Ensign, Scott Choy, Carter Stevens, Emily Wiener, Naomi, and especially my sister Angela. You have been an inspiration, helped me develop personally and professionally, given me opportunities for growth, helped frame my graduate career, and I am forever grateful.

And to all of my friends and colleagues, both past and current. Although I cannot list every influential person that has made a positive impact on my progress through graduate school, there have been a number of key individuals who have been been important to my overall graduate career success at various steps along the way. Including Chris Sheesley, Chris Stivers, Stacey Cha, Nathan Fisher, Katy Davis, Tommy Fisher, Amanda Blauvelt, Alex Trembath, Alex Davis, Pádraig Ó Laoghaire, Ryan Scura, Benny Johnson, Bryan Ting, Ivan Sack, Mitch Westwood, Naiomi Lundman, Mark Noviski, Henry Romeo, Patrick Canfield, Henna Merchant, Andrew Garcia, Melanie Colburn, Tim Strickland, Justin Lam, Marianne Braun, Luis Grimaldo, Alli Shapiro, Dallas Van Wyk, Kanchan Swaroop, Cody Zeger, Lauren Parker, Alexandra Higgins, Erin Huie, Richard Dweck, Patrick Doring, Louise Wo, Troy Nineeteen, Diane Ho, John Savage, among others in my November Project family, my dance community, and other supportive communities I've been fortunate to be a part of in the bay area and beyond. You have provided a welcoming global perspective, encouragement, and support on a variety of pursuits and topics that have enriched my graduate studies. Early morning exercise, hiking, biking, camping, backpacking, traveling, running races, and encouraging me to go beyond my comfort zones and to continue to pursue ambitious goals.

Thank you.

Chapter 1

Introduction

1.1 Structure of this Dissertation

This dissertation is focused on work related to the second half of the author's Ph.D. studies, and focuses on the fundamental combustion characteristics of flames propagating through stratification layers and some aspects of flames propagating through applied electric fields. This first chapter outlines the motivations and possible implications of this work. In Chapter 2, a background of combustion and relevant theory is provided to aid in understanding the implications of this work. Chapter 3 outlines the experimental methodology, experimental setup, and a brief overview of relevant procedures used to perform this work. Then, armed with the necessary background, Chapter 4 goes through the extensive post-processing techniques used to determine the relevant combustion characteristics. Chapter 5 discusses a variety of numerical frameworks used to better understand and interpret the experimental findings. Then, Chapter 6 outlines the main experimental and numerical findings of this work which is then discussed in more depth in Chapter 7. Final conclusions and projections for future work are outlined in Chapter 8.

1.2 Current and Future Energy and Power Demands

Energy and more specifically power demands have been increasing since the industrial revolution. The dependence and need for continued accessible and reliable power has only increased with advancements in technology. This reliable power has been afforded to most developed countries due to the supply of consistent power production. Historically, the security of a nation's energy has dependent on combustion to provide means to meet the demand with investments made to ensure a constant steady power source. This may be why energy supplying the power for developing countries comes from fossil based sources hovers around 80%-85%, while less developed countries rely on 80% of their energy for power from renewable sources [57]. However, those countries that rely on renewable sources do not generally have the infrastructure to balance the intermittent production that generally comes with using renewable energy sources such as solar and wind. Other sources of renewables, such as hydroelectric, geothermal and tidal sources are very specific to regions and are therefore not widely accessible.

In contrast to developing societies with intermittent or non-existent supply of power, a stable supply of power has allowed for comfort and advanced technology to flourish. In doing so, stable energy sources and continued power supply can be correlated with socio-political stability of a nation, as may be inferred in the UN report among others.

The developed world energy makeup is quite similar to the global energy supply, as the Organisation for Economic Cooperation and Development (OECD) has a slightly modified energy distribution makeup as seen in Figure 1.1,

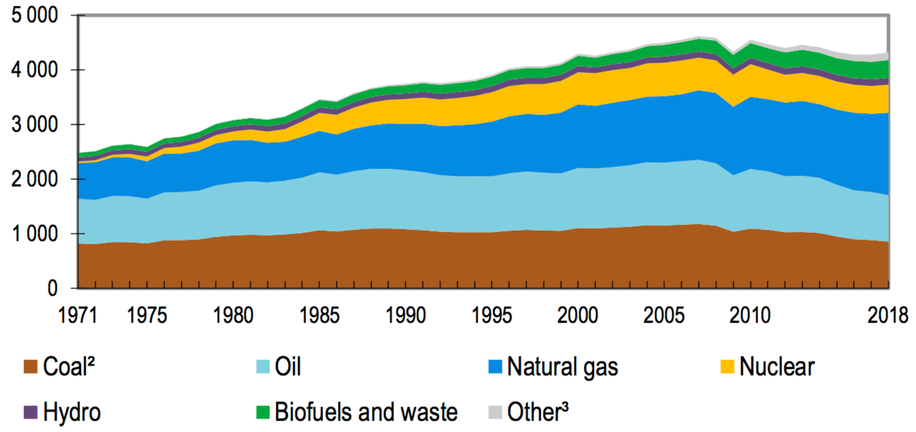


Figure 1.1: OECD Total Primary Energy Supply normalized to Tonne of oil equivalent, Source: IEA (2018) World Energy Outlook[45]

However, since the majority of the power supply to the energy grid comes from fossil based fuels, it has lead to an increasing thirst for fossil fuels that have left severe damage to the environment, including on the surface of the earth and in the atmosphere, leading to pollution and the warming of the earth globally. The choice of fossil based fuel supply, while cheap and accessible, is in fact a slow and steady drain of limited natural resources. As the UN report highlights, increasing the renewable energy sources in the global mix is a priority for the planet. At the same time, the UN recognizes two socio-political needs across the globe, namely the need for stable, reliable and affordable energy and for maintaining and increasing the accessibility to clean water and sanitation [45]. These two goals are coupled as electricity in the developed nations provides a significant portion of the water treatment and sanitation.

The overall attainment of all three of these goals often is in competition, leading nations to choose. In the developed world, life saving devices in homes and hospitals, water treatment and distribution, clothing manufacture and distribution, to food production and transportation, the availability of energy for power has become intertwined with basic societal needs. As power production is a critical infrastructure need, most developed nations prioritize both the reliability and the cost. A leader among the developed nations, the US makeup of energy sources is currently and projected far into the future to be fossil based as can be seen in an excerpt from the US Energy Outlook report in Figure 1.2.

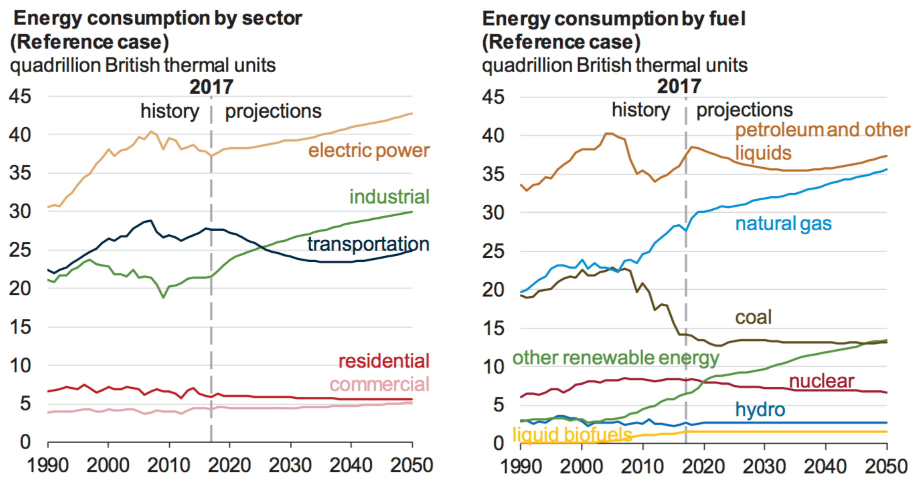


Figure 1.2: U.S.A. Energy Consumption by Sector and Fuel. Source: U.S.A. EIA AEO 2018 [58]

As can be seen from the report, electric power will rise as will industrial and transportation uses, with petroleum and natural gas projected to continue to make up most demand. The lagging renewable sources is indicative of the technical, financial, and political forces still driving this developed nation to choose stability over climate effects.

There are significant resources devoted to maintaining a high standard of living among those able to secure the resources. From air conditioning & heating, water heating, and independence in mobility from automobile transportation to a high selection of energy intensive meat consumption, the exceedingly high standard of living necessitates a high demand on the power grid and on raw materials derived from fossil based sources.

For instance, the end use residential power use among residences in the United States, Pacific Region, and California are compared in Figure 1.3, However, this does not include power necessary for basic infrastructure such as water supply, sanitation, emergency services, among others.

Like the basic infrastructure, numerous goods and services in developed countries are often taken for granted by the end users and this is confounded by the difficulty in determining their impact both on energy consumption and on the pollution. For instance, in a surprising recent report from Quantis that extensively investigated and analyzed the global impact of the apparel industry found that this industry alone produces about 8% of the global CO_2 when the entire value chain is taken into account[42]. Even though this report did not highlight the specific energy used through the industry, the CO_2 emissions alone can

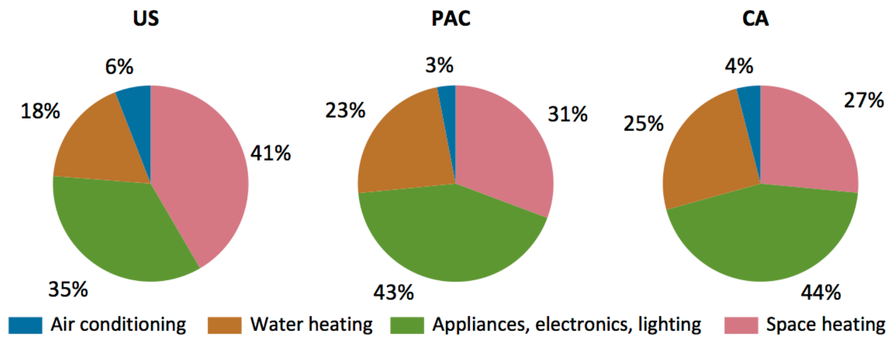


Figure 1.3: United States, Pacific Region, and California, Source: EIA's Household Energy Use in California[18]

be used as a proxy for the energy costs. As highlighted in the report, some of the intricacies of the industry including the variety of partners and supply chains obfuscate the impact of both the energy use and the impact on the environment.

This increase in power demand increased with the computer and technology revolution. High energy use technology services such as computational clusters and data centers, energy intensive software codes such as cryptocurrency, as well as the increase in and high energy consuming electronics at home and at work, all have contributed and will continue to contribute to high power demands. With the advent of exceedingly high power technology such as plug-in vehicles and higher energy loads to combat extreme weather exacerbated by the continued climate change, the acceleration of energy use for reliable power is expected to continue to rise, despite advanced in efficiency.

Current and Near Future Power Production

In terms of suppling power for use, a critical consideration to create a stable energy grid is often overlooked. Electricity must be used within seconds of production or storage is required. The supply of power interplays with the demand of its use, and if too much power is supplied at the wrong time or too much power is demanded then the delicate balance of the energy grid is at risk. Historically, this balance on the grid has been maintained by tight control of the generation of power as it has been quite a formidable challenge to coordinate across the variety of end users. Whether a low technological solution such as turning units on and off, to more sophisticated technology that allows for high turn down ratios to better able to follow the power demand. Industrial scale engineered storage methods have not proved to be economically or technologically viable, yet research is ongoing. Although, one could argue that nature has already developed the perfect storage method, within chemical bonds of hydrocarbons and harnessed in combustion applications.

However most renewable sources have power production rates that cannot be finely controlled to meet the instantaneous demand, let alone guarantee a steady supply. Their introduction into the electrical grid and the necessary compensating control by other units provides new challenges that is a topic of ongoing research. Many renewables by themselves do not support a reliable grid. Whether there is cloud cover, disruptions in wind patterns, or other fluctuating or uncontrollable key component to the production, the power must be wasted or somehow stored until it is used. For renewables this storage may have a much longer time horizon than current infrastructure allows. There are huge limitations to current energy storage techniques, and active research is underway as the costly storage system will undergo frequent and heavy cycles.

With the current push for renewable energy sources and alternative supplies for power production capabilities to meet society's energy demands while reduce global warming and the effects of global climate change, renewed interest in more flexible uses of renewable fuels and technology is necessary. However, there is no silver bullet even if we wanted to go all in, so there is needed time to research and transition to a fully renewable energy supply. Our current renewable technology today has drawbacks to high implementation. Severely high financial costs and risks with the uncertainty of the technology and market, lasting poisoning of the environment with heavy metals and other harmful chemicals in the production of many renewable technologies, and the lack of energy production control necessitates research into additional methods and control schemes to more smoothly and intelligently transition into them and develop and deploy at scale in a sustainable way.

1.3 A Path to Transition

While needed technology is being developed, current infrastructure can be modified to use a different fuel to assist with the transition and provide a control mechanism to match power production with the demand. Ideally this fuel will not require additional infrastructure costs, increased hazards, or significant additional training that would impede or compete with the development of renewable technologies. Even better if this fuel is part of the long term solution.

One such fuel that is currently available for modeling and development as well as has future renewable production potential is methane. Naturally occurring and easily commercially producible, methane is a stable and relatively safe fuel. Methane has numerous sources that include current fossil derived sources as well as numerous renewable sources. The multi-sources are significant in that technology in both the production and use of methane can be performed simultaneously, allowing for faster deployment of this needed renewable technology.

Safety and storage of fuels is a critical factor in deciding which direction to move into. On this front, Methane is a clear competitor. Methane is chemically stable, has a higher energy density and easier storage and safety considerations than hydrogen gas, a lagging competitor. Methane is the main component of natural gas, and it can be produced with high efficiency as biogas, as well as synthesized from hydrogen and carbon dioxide. The methanization process has been well studied and can be achieved in anaerobic digestion from a variety of feedstocks, and it is already a key component to renewable sources. Synthesizing methane has the advantage of allowing for longer term storage than possible with hydrogen, and without the long term health and environmental hazards and waste of batteries.

Fortunately the combustion of methane has been under study for decades with numerous models and tools to predict the behavior in homogeneous premixed combustion environments. Methane heavy fuels have in fact been a key component to contemporary grid stabilization, with the use of natural gas powered plans providing the main backbone to power plants meeting the varying power demand loads. Further integration of methane will assist in stabilizing grids, and accurate modeling of methane's combustion characteristics is essential to advanced designs.

These combustion models assist in the simulation and control of combustion processes and are integral to the engineering design of safe, efficient, robust, and reliable engineering systems. As design modeling will be inaccurate without precise predicts of the burning characteristics of methane leading to suboptimal or flawed designs, efficient models must be evaluated and their accuracy confirmed. While much research has gone into methane containing fuels, much work is still needed. Recent studies have indicated that some of the models may need reevaluation to more optimally predict combustion behavior in stratified mixture configurations.

1.4 Stratification Impact on Current and Future Power Production

Where are stratification layers found?

Stratification layers are ubiquitous and exist both naturally and within a variety of engineering applications. Any time there is a mixing of two miscible fluids, a stratification layer will exist until they form a homogeneous mixture. In designing combustion applications as well as for catastrophic event detective work and safety considerations, the mixing fluids considered are composed of fuel and oxidizer.

In many engineering applications, stratified charges are unavoidable due to limited mixing time and yet it is also often desirable due to efficiency gains, equipment limitations or harmful emission reductions. Common engineering applications where stratification is present include gaseous port and gaseous direct injection into internal combustion engines for various common and lean burn applications, vaporizing liquid from direct liquid injection, as well as the stratification necessary for appropriate temperature control in various gas turbines.

There are also plenty of other applications where a stratified charge may initialize unintentionally either in single layer or multi-layered arrangements. Numerous catastrophic events have occurred in such unintentional stratified environments setup by a variety of phenomenon. Accidental spillage and subsequent vaporizing of liquid fuels such as gasoline or diesel, coal mines gas pockets, overheated nuclear reactors, and wind blowing over a gaseous fuel leak from a pipe or tank are just a few examples. To develop appropriate safeguards to prevent combustion related accidents and others like it from occurring ever again, a deep understanding of the fundamental processes that govern combustion, and more specifically stratified combustion, must be fully understood.

Stratification Research on Current and Future Power Production

This work is focused on the fundamental processes of combustion based power, in particular it strives to improve and further fundamental process knowledge to open and expand additional markets for transition and renewable power production and energy. Additionally, this research strives to better understand the relevant fundamental physical processes that underly current and future research and development of power production methods and safety systems. This fundamental knowledge enables downstream component and system models and simulation to better predict the behavior of combustion processes, including the use of current and alternative fuels for the future.

This advancement in the fundamental knowledge of combustion is primarily realized by a better understanding of flame speed behavior of a methane-air flames transitioning through a concentration gradient under various quiescent conditions. The flame speed behavior is important as it dictates the rate of fuel consumption, and thereby the heat release in combustion applications.

Stratified charge combustion in internal combustion engines, has received some attention over the years for the potential increase in efficiency and reduced emissions. Additional knowledge of stratified flame propagation may aid in the advancement of these technologies for now and the future.

Chapter 2

Background

2.1 Combustion Theory

Mixture Concentration

The method commonly used to identify the concentration mixture of fuel-oxidizer mixtures is to use a non-dimensional number that accounts for varying fuels' chemical makeup. The fuel-to-air ratio is one such non-dimensional number that gives the ratio of the proportions of fuel and air is given by equation 2.1.

$$FA_{mass} = \frac{M_{fuel}}{M_{air}}, FA_{molar} = \frac{N_{fuel}}{N_{air}} \tag{2.1}$$

The ideal stoichiometric concentration for a mixture is one that has the proper composition to follow the ideal irreversible complete chemical reaction equation, converting all of the reactants into products and ignoring the intermediate and reverse reactions. For a methane-air mixture, as in this work, the stoichiometric complete irreversible chemical reaction equation is given by equation 2.2.

$$CH_4 + 2(O_2 + 3.76N_2) \rightarrow CO_2 + 2H_2O + 7.52N_2 \tag{2.2}$$

Therefore the ideal molar concentration expressed as the fuel-to-air ratio is given by equation 2.3.

$$FA_{Stoichiometric} = \frac{1}{2 * 4.76} \tag{2.3}$$

However, to account for differing mixtures, it is customary in many applications to use equivalence ratio. Equivalence ratio is itself a non-dimensional number and a ratio of ratios, the ratio of fuel-to-air of the mixture compared to fuel-to-air of a chemically stoichiometric mixture, in a chemical sense. The equivalence ratio is conventionally a normalized version of the fuel-to-air ratio, namely the mixture value normalized based on the value under stoichiometric conditions, balanced chemical equilibrium, namely equation 2.4.

$$\phi = \frac{FA_{Mixture}}{FA_{Stoichiometric}} \tag{2.4}$$

Either FA_{mass} or FA_{molar} can be used to determine the equivalence ratio, as long as the same choice is used for both the numerator and denominator. However, for the numerical work, and in particular when determining the local equivalence ratio of a reacting flame zone, it is sometimes necessary and convenient to define the equivalence ratio based on atomic masses given by equation 2.5.

$$\phi = \frac{N_{HydrogenAtoms} + 4N_{CarbonAtoms}}{2N_{OxygenAtoms}} \tag{2.5}$$

For premixed unburned mixtures of methane and air, the values are identical.

Flame Surface Speed and Laminar Flame Speed

When observing a spherical flame optically, the most obvious and readily available speed of a flame is the burned gas displacement speed, $V(r,t)^b_d$, of the flame's surface as the radius of the flame grows with time. As the surface of the flame grows from a small kernel to a larger and larger sphere, the flame surface speed changes. This raw change in the surface of the flame can be used to compare the different burning rates of various mixtures when using the same experimental setup. The speed of the flame surface will change depending on the initial thermodynamic properties, such as temperature and pressure. For premixed methane-air flames near ambient conditions, the flame will propagate faster for higher temperatures and lower pressures and slower for lower temperatures and higher pressures.

However, the flame surface speed is not a unique result of intrinsic properties. The value of the flame surface speed directly observed from a flame speed experiment will vary depending on geometric and flow conditions. In addition, the shape and geometry of the flame's surface, the specific geometry of the chamber apparatus, will play a significant role in the flame speed observed. Therefore it is desirable to process the raw flame surface speed and remove the dependence on the flow, shape, and other aspects of the specific flame speed test apparatus for a more readily comparative speed with other testing apparatuses. It is desirable to transform the observed flame surface speed into a quantity that will be more widely applicable, closer to a fundamental property of a mixture that varies only with the mixture, independent of the experimental apparatus used to measure the speed. For premixed flames propagating through stagnant or laminar flow, we call this fundamental flame speed that depends mostly on pressure, temperature and concentration, the mixture's unburned laminar flame speed, S_L.

It is important to first consider what a flame is. A flame is a moving reaction zone, and as such is fundamentally tied to the rate of exothermic chemical reactions. A common first approximation to the chemical reaction rate, $\dot{r}$, of a fuel and oxidizer at a particular temperature, T, is one that follows an Arrhenius rate as given in equation 2.6.

$$\dot{r} = A * [Fuel]^a * [O_2]^b * exp(-\frac{T_a}{T}) \tag{2.6}$$

where A is a pre-exponential factor with first order approximation as a constant, $[Fuel]$ and $[O_2]$ represent the concentration of the fuel and oxygen, and a, b, and T_a are fuel specific exponential factors related to the overall reactivity of the reacting species. It is important to note that the actual chemical reactions that take place in a flame are much more complex with a large number of intermediate species and a tremendous number of intermediate reactions. Typical detailed chemical modeling will use a large subset of reactions and species that accurately represent the overall dynamics in a flame that may not be captured in a simplified equation such as equation 2.6. However, moving forward with the reduced order model, this reaction rate equation can be used to provide a foundation to a first order approximation to an estimate of the laminar flame speed, S_L, namely equation 2.7.

$$S_L = \left(\frac{\dot{r}\alpha(T_b - T_{ig})}{[Fuel](T_{ig} - T_u)} \right)^{0.5} \tag{2.7}$$

Where α is the thermal diffusivity, T_b and T_u are the burned and unburned gas temperatures, and T_{ig} is the temperature at the onset of ignition. While the determination of the laminar flame speed is a bit more complex, equation 2.7 gives a first order approximation of the relevant physical processes on the flame speed.

Premixed Mixture Concentration Influence on Flame Speed

It is well known that the laminar flame speed has a strong dependence on the mixture concentration as can be seen from equation 2.7. The flame speed experiments have shown that the flame speed is highest near the stoichiometric concentration, but slower as premixed mixtures are either fuel rich or fuel lean as can be seen in Figure 2.3. As can also be inferred from both equation 2.6 and 2.7, the flame speed follows the reaction rate and is most sensitive to the temperature. Homogenous premixed flame speeds have been studied and their dependence on both temperature, pressure and concentration are fairly well understood in the context of chemistry and fluid dynamics.

Chemical Considerations

Combustion is characterized by a large number of chemical kinetics and propagating flames are characterized by a self-sustaining reaction carried forward by a number of radial species. As such, concentration of species takes on an even greater importance as molecules compete in a series of collisions of which the collision rate typically modeled statistically depending on the local concentration.

Recipe for a Flame to Propagate

The start of self sustaining combustion requires a set of important endothermic chain initiation chemical reactions to generate a set of important radicals such as those given in equations 2.8 & 2.9 with the molecular collision with a third body species M.

$$CH_4 + M \rightarrow CH_3 + H + M \tag{2.8}$$

$$O2 + M \rightarrow 2O + M \tag{2.9}$$

Once the initial pool of radicals have formed the initial kernel, to be self sustaining the flame must produce more radicals than are absorbed by competing termination reactions. A typical critical branching reaction that will sustain and grow the flame while providing the radicals necessary for a set of exothermic reactions, propagating the reaction front forward is of the form given in equation 2.10.

$$O2 + H \rightarrow O + OH \tag{2.10}$$

However, the formation of additional radicals O and OH are short lived and collide with fuel, oxidizer or other reacting species. There are a number of terminating reactions that will effectively reduce the number of radicals and if the rate of initiating and branching reactions are not sufficient to maintain the radical pool, extinguish the flame.

Pollutant Emissions

Pollutant emissions are formed either with incomplete combustion, as in the case of formation of unburnt hydrocarbons and carbon monoxide

Carbon monoxide is an intermediate species that forms when there is not enough oxygen or the flame is quenched below a temperature of.

The wet route can be modeled with a reaction of the form given in Equation 2.11, with a weak temperature dependence and limited by the OH radicals present.

$$CO + OH \rightarrow CO_2 + H \tag{2.11}$$

The dry route can be modeled as a reaction of the form given in Equations 2.12 and 2.13, but require high temperatures on the order of 1,100 K to significantly contribute to the oxidation.

$$CO + O_2 \rightarrow CO_2 + O \tag{2.12}$$

$$CO + O + M \rightarrow CO_2 + M \tag{2.13}$$

Other pollutants such as nitrous oxides form mainly from thermal pathways that are tied to the temperature of burned gas and of the flame. The strong triple bond in nitrogen molecules is difficult to break and requires significant energetic environments to form oxides in measurable concentrations. Two overall routes of NO_x formation in this work are from local regions of high temperature and along the surface of the flame.

The thermal sources of NO_x has a number of pathways that were first described by Zeldovich, and either have relatively high activation energies to initiate or require combustion radicals to form. Representative reactions are of the form given in Equations 2.14, 2.14, and 2.14.

$$N_2 + O \rightarrow NO + N \tag{2.14}$$

$$N + O_2 \rightarrow NO + O \tag{2.15}$$

$$N + OH \rightarrow NO + H \tag{2.16}$$

The other significant source of NO_x formation occur at the surface of the flame as first described by Fenimore and commonly referred to as prompt NO pathways. These reactions are due to first interaction with CH radicals that form intermediate species HCN and N which can react with other species to form NO. The overall global unbalanced interaction may be represented by the form given in equation 2.17.

$$CH + N_2 \rightarrow HCN + N \rightarrow ... \rightarrow NO \tag{2.17}$$

Flammability Limits

An extension of these ideas leads into the flammability limits of a fuel in air. Intuitively the flame will burn in some mixtures while not in others, and will depend on a variety of factors related to chemistry and heat transfer effects.

One pathway for a flame to extinguish is from an overwhelming number of chain termination reactions that reduces the pool of radicals to unsustainable levels. If the concentration of fuel is too high and radicals find it difficult to react with oxygen, the reactions will eventually consume the effectively available oxygen and terminate the flame. Conversely, if the concentration of fuel is too low, the radicals will find it difficult to react and reactions will eventually be consumed into less reactive species, terminating the reaction.

The other avenue of flame extinguishment is due to the Arrhenius nature of the reactions. The temperature dependence of the reactions results in a reduction in the radical pool and thereby a possible extinguishment of the flame. Although chemistry plays a large role in this pathway, temperature dropping from heat transfer is the leading cause of a second avenue of flame extinguishment.

Flame Stretch

One of the geometric and flow effects that influences the flame is flame stretch. If a flame is propagating through a moving or expanding fluid mixture, flame stretch may exist. Flame stretch can significantly increase or decrease the flame surface speed, and has such a strong influence on the flame that with a high enough flame stretch, the flame can be extinguished entirely. This expansion may be from characteristics inherent in the flow or even from the thermal expansion of the gas from the exothermic chemical processes due to the flame itself.

Flame stretch, a, is more technically given as the normalized change in the rate of area of a differentially small flame surface with respect to time as given in equation 2.18.

$$a(r,t) = \frac{1}{A(r,t)} \frac{dA(r,t)}{dt} \tag{2.18}$$

For spherical flames in particular, flame stretch plays an influential role as the flame is expanding. The flame stretch for such a case is proportional to the inverse of the radius, and so decreases as the flame propagates outward. Flame stretch for premixed laminar spherical flames can be determined based on the radius of the perfectly spherical flame surface, and therefore corrected for to eliminate the effect on the flame speed. For a spherically expanding flame, flame stretch is related to the burned stretched flame speed, V_d^b, by equation 2.19.

$$a(r,t) = \frac{2V(r,t)_d^b}{r} \tag{2.19}$$

The relation allows for a correction for stretch effects on a spherically expanding flame, giving us the unstretched burned laminar flame speed. Both linear and non-linear extrapolation methods have been used to relate the stretched and unstretched displacement speeds, $V(r,t)_d^b$ and $V_o(r,t)_d^b$ respectively. Equation 2.20 gives the linear equation used for a simple extrapolation and Equation 2.21, as used and outlined by Bechtold[3], Kelly[32], Chen[9] et. al, gives the non-linear equation used for extrapolation.

$$V(r,t)_d^b = V_o(r,t)_d^b - L_b a(r,t) \tag{2.20}$$

$$ln(V(r,t)_d^b) = ln(V_o(r,t)_d^b) - \frac{2L_b V_o(r,t)_d^b}{rV(r,t)_d^b} \tag{2.21}$$

where L_b is the Markstein length.

For this work, extrapolation using the non-linear method is used. However, neither of these equations will give the unburned laminar flame speed, S_L, as expansion effects must be accounted for.

Spherical Expansion Displacement Speed

Besides flame stretch effects, outward propagating premixed laminar flames are subjected to expansion effects of the burned mixture pushing the fluid further. Luckily, Giannakopoulos [23] has proposed a way to account for this expansion effect relating the burned unstretched displacement speed to the unburned unstretched displacement speed, $V_o(r,t)_d^b$, based on the density differences of the burned and unburned gases, as given by Equation 2.22.

$$V(r,t)_d^{ub} = \frac{\rho_b}{\rho_{ub}} V(r,t)_d^b \tag{2.22}$$

Relation 2.22 can then be used with 2.19 to relate the raw displacement speed with the true unburned laminar flame speed, S_L.

Stratification Layer and Stratified Flames

Not all gaseous mixtures are homogenous perfect premixed mixtures, and many are subject to or undergo mixing. Such mixing may be due to concentration gradients that are present within the mixture. These gradients in the concentration of differing combustible mixtures are what make up a stratification layer.

There are many ways a stratification layer may exist. Stratification layers are present any time there is gaseous diffusion of fuel and/or oxidizer. One such case can be imagined as depicted in Figure 2.1 where a fuel concentration is graphed above a pool of evaporating liquid, such as gasoline, mixing with ambient air from the atmosphere.

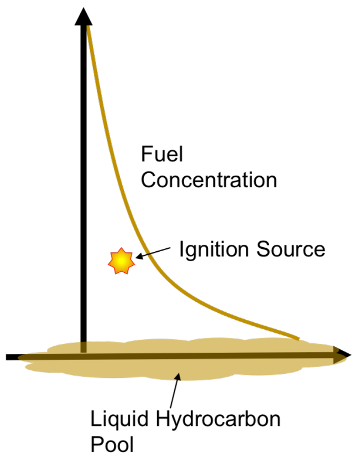


Figure 2.1: Fuel Concentration of Liquid Hydrocarbon Pool Evaporating in Air.

The gaseous concentration will vary from highly fuel rich close to the liquid pool's surface, dropping off to fuel lean above the surface as you move away from the liquid pool and move closer to the atmosphere full of air. Ignition and subsequent flame propagation in the direction of changing concentration occurring within a concentration gradient layer such as this is considered a stratified flame. Stratification layers are also present in numerous other applications, many of which are actively being researched. One such example is gaseous injection systems, where a fuel rich stream is mixing with another fuel lean mixture.

Stratified flames are chemical reaction zones that propagate through a stratification layer. The dynamics of these flames may deviate from homogenous premixed flame speeds as the flame propagates though the mixture, heavily influenced by the dynamic changes in concentration.

2.2 Schlieren Imaging Theory

Schlieren imaging is an optical technique used to observe density gradients in a transparent fluid by use of the differences in the index of refraction, modifying the trajectories of light rays as they passes through different index of refractions. There are numerous optical layouts to achieve Schlieren imaging, but a common low error configuration is a Z-configuration using spherical mirrors. A Z-configuration with spherical mirrors, as used in this work, is depicted in Figure 2.2.

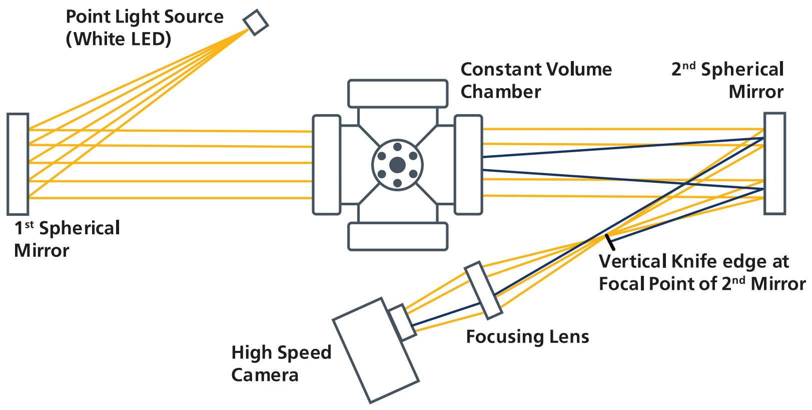


Figure 2.2: Top down view of Schlieren experimental setup with Z-configuration using spherical mirrors and vertical knife edge.

Light is emitted from an LED light source and reflected off a spherical mirror, collimating the light into parallel rays, before passing though the experimental observation volume, the constant volume chamber. Differing densities have slight differences in the optical properties and specifically the index of refraction and will distort the parallel rays of light as they pass though the observation volume. These rays of light exit the chamber and are reflected off a second spherical mirror, where they are focused down to a point where a knife edge cuts off half of the rays of light that have deviated from the differences in the index of refraction. The light rays that are not blocked by the knife edge pass though into a high speed camera and images are captured into frames of a video.

The gradient in the index of refraction is of interest as it can be related to the gradient of the density. Index of refraction can be related to the density by the Gladstone-Dale coefficient, G, as given in equation 2.23. For air the Gladstone-Dale coefficient is about $0.23cm^3/g$ [53].

$$n - 1 = G\rho \Rightarrow G\frac{\partial \rho}{\partial x} = \frac{\partial n}{\partial x} \tag{2.23}$$

Since the deflected light rays due to the difference in the index of refraction are highlighted as light and dark pixels on each Schlieren image, density differences can be correlated. A propagating flame will have the sharpest density gradient near the flame surface as the unburned gas reacts. Therefore, the location of the flame surface can be identified from the sharp gradient in density from each image indicated by the dark and light pixel surfaces.

2.3 Fluid Dynamics

As a flame propagates through a fluid, it is important to consider the coupled interaction of combustion and fluid dynamics. There arises a few relevant instability dynamics in this work that arise from a shearing layer and unequal diffusion of energy and mass. As also highlighted by Law[34] and assembled by Pelcé[15], early experimental and theoretical work on hydrodynamic instability of flames by Darrieus[13], Landa[33], Markstein [36][37], and Einbinder [17] et. al developed the early framework in flame instabilities. Their early work and theory laid down the framework for both thermo-diffusive instabilities as well as hydrodynamic instabilities.

Thermo-diffusive instabilities

Thermo-diffusive instabilities occur when there is a unequal molecular diffusion compared to thermal diffusion under flame stretch. To gauge the relevance of this type of instability in a flame, the Lewis number, Le, may be determined in the region of interest. The Lewis number, defined as the ratio of the simultaneous heat and mass transfer by convection, namely equation 2.24.

$$Le = \frac{\alpha}{D} \tag{2.24}$$

where α is the thermal diffusivity, $\alpha = \frac{k}{\rho c_p}$, and D is the mass diffusivity present in Fick's law of diffusion. As such, the Le number depends on the fluid, the flow, and the species present.

It has been shown that when $Le < 1$, instabilities may be generated in the flow. Small molecules are usually the only species that can move fast enough and penetrate further than their ability to transfer heat. While low conducting gases can also exhibit such behavior, typically the presence of hydrogen species in a flame is the only type of flame to exhibit this instability behavior due to hydrogen's small size and fast molecular diffusion within combustion. This type of instability manifests itself typically as a cellular structure in the flame surface.

It is important to note that both Einbinder and Markstein studied flame instabilities and noted that there were various thermo-diffusive instabilities present in a finite thickness flame leading to stabilizing or destabilizing net effects depending on the Le number. In particular, Einbinder[17] noticed that in slow lean flames that propagate by atomic radicals will exhibit small cellular instabilities when they transition to a richer mixture that is dominated by thermal diffusion. This transition leads to a $Le < 1$ that induces instabilities, and for methane-air flames in the form of small cellular structures. Furthermore, Markstein[37] and Behrens[4] found that addition of hydrogen to methane flames lead to cellular structures.

Kelvin-Helmholtz Instabilities

Kelvin-Helmholtz instabilities are hydrodynamic instabilities resulting from velocity shearing between two fluids moving at different velocities. This fluid dynamic effect is typically seen in cresting of ocean waves and is common in weather patterns visualized by characteristic billowing wave pattern also called fluctus cloud patterns. This pattern has been modeled using a phase-field model and shown by Geun and Kim[22] that higher surface tensions or density ratios lead to reduced growth of this instability.

2.4 Past Work in the Literature

Homogenous Premixed Laminar Flames

Numerous past experimental studies have investigated methane-air flames in quiescent chamber environments where propagating flames are observed and measured. Techniques to quantify the flame displacement speed have been used for a number of decades. The use of jet burners[52], et. al. , opposed jet fuel-oxidizer streams [16], et. al. , flat plate burners [14], et. al. , constant volume chambers and constant pressure chambers and channels [35], [59], [56], [50], [40], et. al, have all used pressure and or optical means to measure gaseous methane-air flame displacement speeds with reasonable success.

Reasonable yet not precise agreement on the laminar flame speed has been achieved between the different methods. Figure 2.3, gives a selection of extrapolated values of unburned laminar flame speed given in the literature for varying equivalence ratios.

Discrepancies in the literature from spherical premixed flame measurements in particular have been analyzed and the sources of error and uncertainty have been well documented. The main sources of deviation in flame speed are outlined by Chen[8] and can be attributed to deviations in equivalence ratio, hydrodynamic effects, and numerical extrapolation, among other reasons. Equivalence ratio control is paramount given the high sensitivity of flame speed to equivalence ratio, however, temperature and pressure deviations in mixture equivalence ratio preparation can result in deviations in the equivalence ratio and in turn, in the flame speed. Additionally, buoyancy effects can influence slower flames and can be neglected when the laminar flame speed is more than 15 cm/s. While thermo-diffusive instabilities may not be typically present in methane-air flames, the flame may develop hydrodynamic instabilities. Additionally confinement of the flame that raises the pressure and radiation

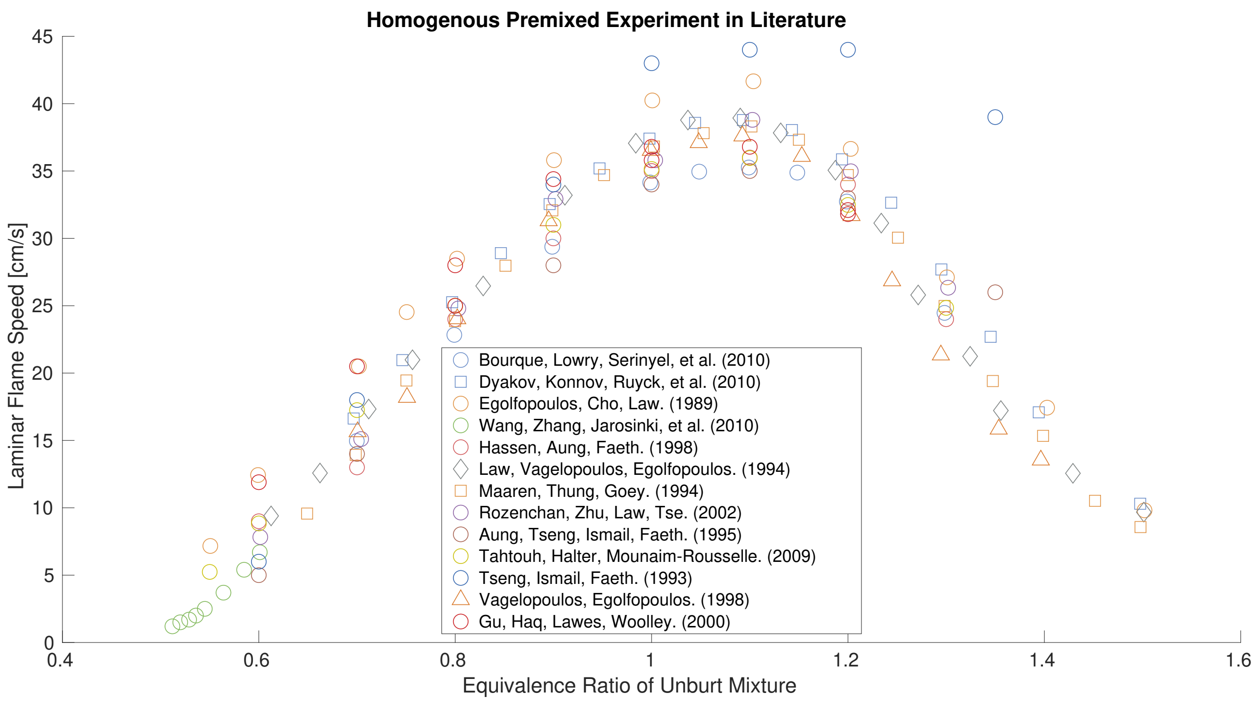


Figure 2.3: Laminar flame speeds in the literature for methane-air premixed mixtures near ambient conditions as a function of equivalence ratio.

losses can both affect the development of the flame and lead to deviations in the laminar flame speed determined between different experimental apparatuses. In particular, radiation has been shown to decrease the flame speed for spherical flames far from stoichiometric [7].

Finally, there are a number of techniques in post-processing of the spherical flame data to limit the deviations in the experimental results. Proper selection of the radius to remove ignition effects is key and typical values given in the literature are around 6 mm to 8 mm. Different methods of zero stretch extrapolation have been used and may result in deviations, dependent on the equivalence ratio. For methane-air flames and a non-linear method is recommended for high equivalence ratios that correspond to large Lewis number. Furthermore, the radius chosen for zero-stretch extrapolation can result in deviations in the flame speed.

Building off these and other measurements, models have been derived to estimate the laminar flame speeds for a wide variety of mixtures. Countless numerical models have used these estimates as well as chemistry data to build chemical kinetic mechanisms to model the complex interactions of chemistry and flame dynamics. A heavily validated chemical kinetic mechanism, the 53 species GRI 3.0[55] natural gas mechanism, is commonly used to model methane-air combustion. This mechanism was used in this work for methane-air combustion numerical models.

Stratified Flames

Significantly less experimental and numerical work has been done in the past compared with homogenous premixed flames. The limited experimental work may be due to the extreme difficulty in controlling the physical setup of stratified mixtures.

Early experiments by Karim and Tsang[31] using a vertical quartz circular tube of 9.5 cm (3.75 in) inner diameter with a plate to separate two methane-air mixtures and high speed photography to detect the initial flame speed enhancements. Their work utilized a theoretical model to determine the fuel concentration profiles along the flame path from two homogenous premixed mixtures. The plate was removed and the gases were mixed by the induced convective motion of the plate as well as natural inter-molecular mixing a few min to hours before ignition, and noted the flame propagation deviated from predicted homogeneous values.

Girard, Huneau, Rabasse and Leyer[24] studied two nearly concentric hemispherical soap bubbles of radius 15mm to 30 mm (0.59 in to 1.18 in) made of aqueous sodium oleate separating mixtures of propane-air and hydrogen-air for stratified mixtures between stoichiometric and lean, detecting the flame front speed from Schlieren images. They noted that the flame either became cellular or turbulent as the flame passed through eddies induced by the soap bubble popping, and that any flame they looked at could be accelerated by the eddies induced by the popping of a bubble. Despite this, they noted qualitative enhancements that depended on the stratification direction with higher speeds in the rich to lean case, and a slight expansion of the lower flammability limit of hydrogen. In addition, Girard et. al noted in a theoretical model of the pressure fields that the stoichiometric to lean case had a different burning and pressure profile than the lean to stoichiometric case.

Badr and Karim[1] continued the semi-planar flame propagation in a constant pressure 63.5 mm (2.5 in) diameter circular Plexiglass tube with a plate to separate two methane-air mixtures, looking at different tube orientations and measured the flame propagation speed from light detectors located along the tube. Due to the setup, the flame responded to pressure oscillations from the semi-closed end on the order of the diameter of the tube. They placed heavier mixtures above lean mixtures to enhance the intermolecular diffusion, and concentrations were determined theoretically and checked using sonic transducers. Speed ups of stoichiometric to rich and stoichiometric to lean were noted while rich to stoichiometric and lean to stoichiometric showed no significant difference in speed, and differences in flame spread were attributed only to the mixture stratification when compared to homogeneous cases. They also noted that flame front instabilities in their work were enhanced with stratified mixtures, and therein the velocities of the flame fronts.

Continuing that work, Karim and Lam[30] focused on the 63.5 mm (2.5 in) diameter vertical tube separating methane below and air above with a more detailed concentration profiling using an ultrasonic pressure transducer and a sparking system along different points of the tube. A section of their work was on profiling the concentration gradient and indicated that significant diffusion of the mixtures happened near the interface in less than a second and longer than 30 min were required to reach a steady concentration profile. Ignition was achieved at the leaner mixtures below that reached with homogeneous mixtures and attributed to the stratification and mixing. They noted disturbances in the flame front, but attributed it solely based on the pressure disturbances from the closed tube. Enhancements were noted for flames traveling from the lean limit to stoichiometric and attributed to the enhanced convective mixing ahead of the flame, disrupting the expected local concentration.Slow downs were also noted in stoichiometric to rich stratified flames, but were faster than their determined homogeneous counterpart flames which was attributed to preheating of the unburned mixture by the flame from conductive and radiative processes.

Preliminary work and development of a method to investigate igniting lean burning zones and the effect on emissions, Furuno et. al[19] developed a method to ignite a flame and propagate it through into a lean mixture using a soap bubble to separate the mixtures. In this work, they prepared lean mixtures, ϕ=0.6 and 0.7, and richer mixtures, ϕ=0.8, 1.0, and 1.2, within tanks equipped with agitators and allowed to premix for 12 hours. The use of a undisclosed soap mixture composed mainly of a surfactant and Glycerol was used to make the NO_x emission optimal 7.5mm (0.295 in) in radius bubble separation. Initially they tested homogenous mixtures with the bubble and they noted that THC was higher by about 10% for bubble cases, but that NO_x emissions did not increase appreciably for the smaller bubble sizes tested. They noted an overall improvement of 39% in the overall combustion time, indicating a faster flame, for the stoichiometric to lean case. Although Schlieren imaging was used in this set of experiments, there was no discussion of the flame surface.

Using the method developed by Furuno et. al to ignite a flame, Ra[44] looked at pressure rise measurements within a sealed 76.2 mm (3 in) radius spherical constant volume chamber using a 3.175 mm (0.125 in) tube to form a bubble of radius 7.5 mm (0.295 in) to separate two mixtures from rich to lean ignited by a Nd-YAG laser were noted to accelerate the flame. However, in this study Ra used Schlieren imaging to observe the ignition kernel and the initial flame development and noticed wrinkling of the flame at interface region of the two mixtures. The flame surface was neglected in the analysis as the wrinkling dissipated and was attributed to the bubble and the quantity of soap solution used run by run. Homogenous mixtures were made in a mixing tank using the partial pressure method, but were not noted to be validated. The mixture stratification setup was assumed an idealized step function in theoretical modeling. The bubble's ability to contain the mixture was checked experimentally by noting that the slope of the pressure rise from a stoichiometric mixture within the bubble and air outside did not appreciably change when different bubble sizes were blown. An aberration was noted in the pressure trace and attributed to the flame reaching the bubble, but no additional evaluation of the flame surface through propagation was made.

Later, Kang and Kyritsis[28][29] used a novel isobaric 2 cm by 2 cm (0.78 in by 0.78 in) square channel fed by two gas streams above, a mixture of methane and air, and below, air, to measure the semi-planar downward flame propagation speed through a stratification layer. The mixture stratification was modeled through a one dimensional theoretical diffusion relation and measured by the methane-air stream doped with acetone and PLIF measurements to determine the concentration by assuming acetone perfectly mixed with methane and indicative of mixture concentration. One dimensional concentration was determined through a curve fit of the PLIF data, with noise in the equivalence ratio was attributed to image gain of the camera used. Using a Schlieren with a high speed camera they noted higher than adiabatic flame speeds and an extension of the flammability limit. With the assumptions and models employed, they noted flame propagated beyond the flammability limit. In addition, they concluded that there was a holistic and integrated dependence on the flame history and not just the local equivalence ratio or the local equivalence ratio gradient alone. Little comment on the flame structure was made and assumed relatively planar, and flame speeds were noted to increase by a factor of two compared to homogenous values.

In a different setup, an oblique stratified methane-air flame was studied extensively by Galizzi and Escudié[20], where they setup a flame front stabilized on a 2 mm (0.0787 in) diameter rod on the exit of a wind tunnel operating at 5 m/s and 2 mm upstream a rail injected pure methane to setup the mixture and stratification of both lean to rich and rich to lean in the same setup. Mixture concentration, flame topology, combustion characteristics and temperature were measured extensively through the use of Particle Image Velocimetry (PIV), laser tomography, CH^* chemiluminescence, Laser Doppler Velocimetry (LDA), thermocouple measurements, and Non-Dispersive Infrared (NDIR) for CH_4 concentration. They noted a wrinkling of the stratified flame on the order of the stratification thickness with a periodic structure even though both the flame and the flow were laminar and attributed the wrinkling to instabilities created in the shear layer at the burner edge. In this region they noted a widening of the flame front with the flame propagating faster into the stratified mixture. This stratification induced a disturbance in the flow that disrupted the symmetry of the flame, affecting the upstream flow nature including propagating into the upstream mixture, and disrupted the ability to fully measure flame speed through the stratification layer. The wrinkling structure they noted to be attributed to various velocity disturbances and were unable to fully describe the origins but noted that the process took approximately 0.5 ms to form and grew regularly. Additionally, within the stratified mixture they noted a diffusion front due to higher CH* radicals downstream of the flame front and as the flame propagates into a richer mixture.

Experimental and numerical work by Schmidt and Kyritsis[51] on methane-air and propane-air stratified flames over a flat plate burner with stratification generated by changing the incoming mixture stream and visualized with chemiluminescence. They indicated that flames burning through stratified mixtures underwent local stretch effects and some reduced mechanisms may be unable to accurately capture the chemistry resulting in additional instabilities. The chemiluminescence photographs indicated a wrinkled propane flame and an even further wrinkled methane flame. They discussed how slight disturbance in homogeneous flames amplify disturbances, and the wrinkled flame structure in their images was noted to be disturbed but attributed to limitations of the flow field experimental setup.

Preheated (300°C) premixed and stratified oscillating opposed flow methane-air flames were studied by Cuoci, Frassoldati, Faravelli, and Ranzi[12] in a 14 mm (0.551 in) counterflow burner with 0-3% hydrogen addition with a separation of 7mm (.275 in) and acoustic speakers provided oscillations and Laser Doppler Anemometry measured the velocity profile. They indicated that the back support of a stratified mixture could support the flame through stretch extinction limits normally allowed a steady flame. It was noted that a partial stratification was induced by the oscillating flow with an accumulation of methane in the stagnation plane and a enlargement of the flame zone. The flame structure with the addition of hydrogen was noted to not change compared to the no hydrogen case, and hydrogen was able to extend the strain rate of oscillating flame further than originally capable with contribution related to its influence on the flow field.

Balusamy Cessou and Lecordier[2] looked at propane-air stratified mixtures using a impinging laminar 5 mm (1.96 in) jet to introduce a mixture stratification in a 2 cm by 2 cm by 16 cm (0.787 in by 0.787 in by 6.299 in) constant volume chamber with quartz windows. PIV-PLIF measurements with seeded Anisole were used to deduce the local equivalence ratio to about half a millimeter and the flame speed. The mixture is setup using a mass flow meter and a sonic orifice plate. The flame was subject to additional stretches not present in a spherical flame, and it was noted a qualitative enhancement into the lean mixture was achieved when burning from a rich region. No flame structure analysis was performed but it was noted that disturbances could arise from differing stretch and equivalence ratio expansion. In addition, some small ripples were present in the presented images of instantaneous flow and scalar equivalence ratio field.

All of the above mentioned small-scale stratified flame experiments have helped with the understanding of stratification. In support of numerous theoretical models developed by these experimental teams, past numerical work has highlighted possible mechanism of the effect of stratification.

Following experimental work, Westbrook and Chang at Sandia investigated the possibility of stratified flame propagation deviating from homogenous premixed flame propagation.

Later, in support of the experimental work, Ra[44] used the Sandia HCT code to simulate a one dimensional planar flame to account for the faster expansion of stratified flames noted in experiments.

Building off prior work from Ra and the Sandia report, Cruz et. al[41] further investigated a stratified adiabatic methane-air planar flame with the Sandia HCT code and attributed differences in stratified flames to both chemical and mass and heat transfer effects, providing a better understanding of the underlying mechanisms as to why the stratified flame speed did not exhibit premixed flame speed transitions. In particular, they attributed the flame speed up from rich flames providing preferentially diffusing H_2 species when transitioning from rich mixtures to stoichiometric. Conversely, the speed up from stoichiometric to lean flames was attributed less with a preferential diffusion of hydrogen, and more with the additional thermal transport from the burned gases, leading to faster thermal breakdown of the fuel into H and H_2 species. In addition, they noted a rich flame burned into lean burning zone well beyond experimentally determined extinction limits but partially attributed it to the numerical setup and its ability to sustain chemical reactions beyond which are sustainable in an experimental context. Their simulations also demonstrated a memory effect on the flame when burning through a set of thin layered stratified mixture layers. The simulated flame burned from a stoichiometric mixture into a rich mixture before continuing to propagate through a thin layer of air where it was able to continue into a stoichiometric mixture.

Kang and Kyritsis [27] continued this work on the memory effect on methane-air flames using theory to give further backing to the importance of the overall history the stratified flame undergoes.

A numerical model of a methane-air counterflow burner using a part of the CHEMKIN Collection, OPPDIF, was studied by Richardson, Granet, Eyssartier, and Chen[48] where they simulated a flow of reactants opposing a flow of products to investigate equivalence ratio gradient, strain rate effects and steady and oscillatory effects. In particular, they looked at stratified equivalence ratios starting/ending from 0.6 to 0.9. They attributed speed enhancements and flame thickness increase of stoichiometric to lean stratified flames to the diffusion of radical species ahead of the flame, and in particular for back supported flames compared to premixed flames they noted that despite the lower flame temperature at the peak heat release of the stratified case it contained 0.3% higher H_2 species, 5.8% H species, and 6% OH species. Many of these radicals were noted in higher concentrations through the flame front and into the unburned gas, indicating a higher radical flux through the flames in the stratified cases.

Zhou and Hochgreb [64] also investigated the counter flow behavior of reactant opposing products of methane-air stratified equivalence ratio streams with additional stratification equivalence ratios investigated. They concluded similarly as Richardson et. al, that stratified flames from the lean side are dominated by heat support while rich flames dominated by hydrogen preferential diffusion.

Zhang and Abraham [63] studying methane-air and hydrogen-air flame simulations in a one dimensional Flow Large-Eddy Direct-Simulation code. They noted that speed ups of the stoichiometric to lean methane-air stratified flame may be partially attributed to the thickening of the flame as a stratified mixture transitions into a lean mixture.

Later, Shi[54] simulated a spherical stratified flame using a one dimensional adaptive compressible flow code for hydrogen-air, methane-air, and propane-air fuels with a stratification thickness on the order of 1 mm. He noted that methane produces more intermediate hydrogen in rich flames and the flame propagation enhancement of methane-air stratified flames are enhanced by the preferential diffusion of molecular hydrogen. Expanding stratified flame simulations to include a full history effect of a rich mixture, equivalence ratio of 1.6, to a lean mixture, equivalence ratio of 0.6, and indicated the local stratification gradient and equivalence ratio insufficient to capture the flame dynamics.

Even with all of this experimental and numerical work, additional questions remain regarding the physical mechanisms, structure, enhancements and emissions of laminar flames and as they relate to downstream models and applications.

Chapter 3

Experimental Methodology: Apparatus and Relevant Procedures

3.1 Constant Volume Chamber Setup and Data Acquisition

The experimental setup was built with a 1.45 L constant volume chamber that has been used successfully to measure flame, flow, and ignition experiments. The chamber is made from two intersecting stainless steel pipes, welded together and closed off with two quartz windows for optical access, and two aluminum plates and grounded, as depicted in Figure 3.1. Mounted on one of the aluminum plates, a 6052B Kistler piezoelectric high frequency pressure transducer connected to a Type 5010B Kistler amplifier to allow for fast and accurate pressure measurements within the chamber. On the opposite side aluminum plate, a type K thermocouple was installed to measure the unburned gas temperature.

An optical pass-through is afforded by two 11.43 cm (4.5 in) quartz optical windows, with a view circumference of 10.16 cm (4 in) that allows light to penetrate the main chamber experimental area, allowing Schlieren imaging of the flame surface and structure as the flame develops and propagates in the chamber. A Schlieren z-configuration utilizing two 7.62 cm (3 in) spherical mirrors allows for low error flame surface detection visualized from the light deflections due to gas density gradients. A 2.1 MP high speed Photron FASTCAM SA4 camera captured raw full resolution black/white Schlieren images at 6000 frames per second, or 0.166 ms between each frame.

The Schlieren images were used to determine the reaction zone as the flame propagated through 10.16 cm (4 in) in diameter. The flame speeds measured using this imaging method followed a nearly constant pressure as determined by the pressure transducer during the initial expansion viewed in the window, and so pressure effects on the flame speed were deemed negligible.

The chamber is sealed to allow for constant volume experiments, as well as to contain the burned gases for emission testing after a test is performed. Between each run, a vacuum

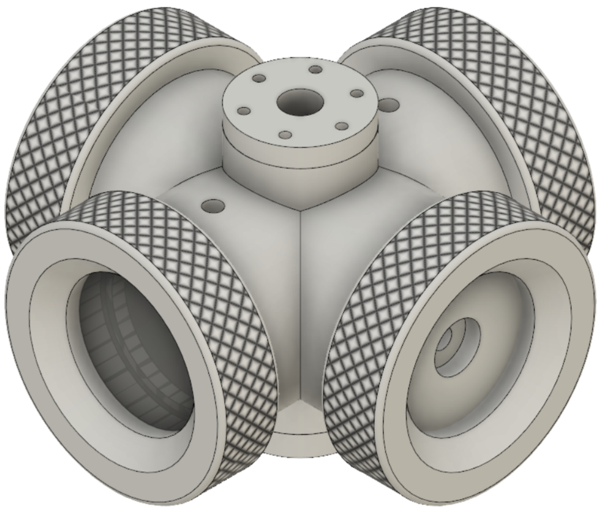


Figure 3.1: Constant Volume Chamber with optical access and side mount for pressure transducer and top hole for stratification and sparking systems.

pump extracted the residual gases through a port in the top of the chamber. A second vacuum line was accessible from within the soap dish pan, and was used to vacuum and purge the straw line before filling. The chamber was filled with mixtures either from a second port on the top of the chamber or from the straw tube used to blow the bubble. This setup allows for the safe experimentation of combustible gas mixtures with measurement of numerous relevant characteristics.

Ignition in the constant volume chamber was employed by the use of a custom set of 0.15875 cm (0.0625 in) diameter pins to provide a consistent pin-pin sparking for development of a spherical flame. Good alignment in the pin-pin ignition system was achieved, and a spark gap of 1.727 mm (0.068 in) was used. The ignition energy was held constant through out the runs and radial measurements were taken away from the initial kernel to eliminate any stray ignition effects. A stock engine sparking coil and a 12V battery was used to provide the ignition energy for each spark. To estimate the spark energy deposited in the chamber to ignite each flame, a multi meter was used to measure the battery voltage and a current clamp was used to measure the sparking coil charging current.

The experiments utilized a custom LabVIEW VI program, shown in Appendix A.1, using a National Instruments USB-6343 DAQ board to provide safe experimental control and data acquisition (DAQ), including syncing of the ignition and high speed camera. The program allowed for changing the spark coil charging time to modify the ignition energy deposited, the ignition timing, and gathered various measurements before and during an experimental run. The DAQ also read in data that was captured from a Tektronix TDS 3024B Four Channel Oscilloscope and saved to a separate output file. The signal from the pressure amplifier was read in simultaneously by the onboard DAQ as well as well as captured from an oscilloscope. Charging current to the spark coil was measured with a Hantek CC-65 AC/DC Current

Clamp set to 1mV/10mA and connected to the oscilloscope. In addition, the spark and camera signals were sent to the oscilloscope to providing timing shifts and align the data.

All of the components and upgrades to the constant volume chamber was designed and manufactured in-house. Assistance was provided by undergraduate volunteers and guidance from shop staff. Post-processing of the high speed camera data was done using a custom code due to the constrains of the Schlieren images, and is further discussed in the following chapter and relevant code is provided in Appendix A.2.

3.2 Homogenous Premixed Mixtures

To setup the stratification layers, the use of controlled and well mixed homogenous premixed mixtures were made and separately tested for validation and benchmarking. Numerous homogenous premixed methane-air mixtures were made in batches and mixed in high pressure tanks. All of the flame experiments utilized a set of the homogenous premixed tank mixtures, and great care was given to ensure the mixture was as accurate and as well mixed as possible.

To reduce the limited variability in mixing tank, the high pressure tanks were filled as high as possible to allow the most relevant comparisons and to check for variability in the tank mixtures. Temperature and pressure within the tanks was limited to ensure safe operation. The minimum auto-ignition temperature of methane-air mixtures in ambient pressure is around 600 C[49] et. al, and so care was taken to avoid temperature fluctuations well below half this temperature to ensure decomposition of the fuel did not occur. Additionally, the maximum pressure of the house air line provided a limit to the maximum pressure of a fully mixed tank that was well below the maximum pressure following an unlikely combustion event within the tank. The maximum pressure of gas was determined safe through two considerations. First a deflagration wave was assumed, giving the maximum pressure rise from complete adiabatic combustion of a full tank at the maximum pressure would not raise the pressure of the gas over the designed pressure limits of the high pressure tank. Second, a detonation wave was deemed impractical as the initiation energy was significantly higher than capable with the setup, with values of approximately 4.7E4 MJ as given by Wolanski[62], et. al. Despite the above points, additional care was taken to ensure no reaction could propagate into the tanks with both procedural and engineering safeguards.

Making the Mixtures: Partial Pressure Tank Mixture Filling

To make specific premixed mixtures, a partial pressure fill technique was used measuring each tank pressure with a high precision 3D Instruments Accu Cal Plus 100 PSI pressure gauge, model 75514-23B55. The procedure to fill each tank followed the general form as outlined below.

First the mixing tank is purged to ensure accurate tank target concentration. To do this, the tank is vacuumed out to remove as much previous gas as possible. Then, to remove residual impurities, the tank is purged by filling it with dry house air. The tank is allowed to mix for a minimum specified time, and then vacuumed out to remove the purging air mixture.

Once cleaned, raw gas filling starts and the tank is filled with a three layer fill technique. The tank is filled first with dry house air to slightly above atmospheric pressure, P^{abs}_{air-1}, where the pressure gauge is most accurate. Subsequently, the fuel, methane, is then added to a target pressure, P^{abs}_{CH4}, slowly and allowed to equalize. Finally, dry house air is added to pressurize the tank to the target pressure, P^{abs}_{air-2}, to reach the desired tank mixture equivalence ratio following the partial pressure equivalence ratio technique.

Partial Pressure Equivalence Ratio Determination

The partial pressure technique provides a way to closely target a specific equivalence ratio. It relies on the idealization that a mixture of ideal gases do not individually interact, thereby the overall pressure, P, of a mixture is a sum of the individual pressures, P_1,P_2, ..., of the individual gases. For air and methane at filling temperatures and pressures in this work, their behavior can be modeled as an ideal gas, which follows the following ideal gas law equation given in equation 3.1.

$$P = \frac{NRT}{V} \tag{3.1}$$

Since the high pressure tank is relatively rigid, the volume does not change noticeably. With an appropriate fill procedure, the mixture temperature will also not change appreciably, and therefore the molar ratio of fuel-to-air can be determined from the pressure ratio alone to determine the concentration of the fill gas.

The equivalence ratio of the mixture can then be determined based on the concentration ratio of fuel-and-air given the intermediate filling pressures of air, P^{abs}_{air-1} and P^{abs}_{air-1}, and intermediate pressure of fuel, P^{abs}_{CH4}. These intermediate pressure values can then be used with equation 3.2 to determine the overall mixture equivalence ratio.

$$FA = \frac{P^{abs}_{CH4} - P^{abs}_{air-1}}{P^{abs}_{air-2} - (P^{abs}_{CH4} - P^{abs}_{air-1})} \tag{3.2}$$

Dividing equation 3.2 by the ideal molar stoichiometric fuel-to-air ratio, equation 2.3, the equivalence ratio can then be determined as given by equation 2.4. Slight corrections were performed to account for such pressure effects as daily fluctuations in ambient pressure.

Tank Settling and Mixing Time Estimate

Ideally the tank mixtures could be agitated or mixed through a controlled diffusion enhancing process. Various methods with the equipment available were attempted. However, limitations in the equipment available limited the extensive use of such external mixing mechanisms with the high pressure tanks used. In addition, since numerous prior experimental work on homogenous premixed flames have used a mixing tank to prepare mixtures of various equivalence ratios, this method was used.

Initially, the tank mixing time was estimated based on past experimental methods presented in the literature after accounting for pressure effects. However, following a variety of tests for the homogeneous premixed tank mixtures, additional care was necessary to ensure proper mixing and limit variability between runs for the same tank mixture.

The estimated minimum time allowed for mixing was determined through theory, and is dominated by the end filling tank pressure and the size of the tank. Since many of the mixtures tested ran close to the flammability limit, the flame surface speed was highly sensitive to the mixture concentration. An appropriate waiting time was used to allow the mixture to thoroughly mix. A dimensional analysis was developed to account for the mixing time, estimating the mixing time, and confirmation through experimentation.

To estimate the comparative time it takes for a gas to mix, a characteristic time can be determined. Using multiple characteristic times can be used to determine adequate mixing, and such a method was used in this work along with subsequent consistency validation experiments. To estimate the time for each time constant and therefore the time to properly mix the tank, gas kinetic theory was used to adjust for each tank mix depending on the tank size and the pressure of the final mixture. Fick's law gives a linear proportional relationship between the diffusive mass flow rate and the concentration gradient, given in equation 3.3.

$$J = -D\frac{\partial C}{\partial z} \tag{3.3}$$

The proportionality constant is called the diffusion coefficient, D with units of m^2/s, and relates the diffusion rate, J with units of kg/s, to the molar concentration gradient. If D is known or can be approximated, the rate of diffusion for a given concentration gradient, $\frac{\partial C}{\partial z}$, can be determined. Therefore, estimating the diffusion coefficient gives some indication of the relative speed at which molecules diffuse.

Gas kinetic theory can be used to estimate the diffusion coefficient for a molecule to diffuse through another gas. For an ideal gas, the diffusion coefficient can be modeled following gas kinetic theory and relationship to temperature and pressure extracted, as given by Warnatz et. al[60],

$$D \propto \frac{T^{\frac{3}{2}}}{P} \tag{3.4}$$

Armed with this relationship, the rate of diffusion dependence on pressure and temperature can be determined.

A non-dimensional characteristic time, τ, can then be determined using equation 3.5.

$$\tau = \frac{L^2}{D} \tag{3.5}$$

The characteristic time depends on a characteristic length, L, which will depend on the length the gas mus travel to mix, and the diffusion coefficient. Using the preceding equations together, if a gas takes 10 seconds to diffuse and mix with another gas at ambient pressure, at 10 times the ambient pressure the same gas molecule will diffuse 10 times slower and take about 10 times as long to diffuse the same amount.

Diffusion coefficients for methane in air at ambient pressure have been determined and summarized in the literature [11], [38], et. al. For this work, it was determined necessary though experimentation and evaluation of the variability of the flame speeds from numerous tank mixtures, that on the order of 10 time constants were needed to ensure sufficient tank mixing and minimize tank variability. With the size of the tanks used and considering the temperature and pressure for safe operation, this resulted in a mixing time on the order of two weeks between filling the tank and for complete stabilization to be ready for use. Multiple methods were attempted utilizing the resources available to decrease the wait time, but none were able to replace time.

3.3 Experimental Setup: Homogenous Premixed Flame Propagation

A series of homogenous premixed experimental runs were performed to provide necessary validation and benchmarking. Homogenous premixed mixtures were burned in the constant volume chamber and the resulting flame speeds were measured over the course of the work. Some gas from each tank used in the stratified experiments were were tested in this way. Following vacuuming the chamber from a port on the top of the chamber, the chamber was filled with a premixed gaseous mixture using a well mixed homogenous high pressure tank through another port on the top of the chamber. The pin-pin setup ignited the spherical flame in the center and allowed to freely propagate while pressure and Schlieren imaging data was collected. After the gas had burned through, gas products were fed into a purged and vacuumed exhaust tank for measurement.

As there is limited exploration performed in mixtures far from stoichiometric, burning homogenous premixed mixtures provides additional flame speed data. This particular experimentation category also provided necessary baseline values of flame speeds to compare later stratification runs to that controls for the specific geometric, setup, and conditions present in this experiment that cannot be accounted for otherwise.

Practically, the homogenous premixed mixture experiments provided a secondary method of verification of each tank mixture on top of the mixture concentration determined from the partial pressure technique. Homogenous premixed flames were a necessary validation tool when procedure improvements or equipment upgrades were made during this work. Lastly, these experiments allowed for a controlled way to refine many of the procedures and initial post-processing in this work.

Validation of Setup

Extensive validation of the homogenous mixtures was undertaken. The equivalence ratios determined through the atmospheric corrected partial pressure measurements were compared with the equivalence ratio determined from gas analysis and using the Brettschneider method[5]. The flame speeds were compared to literature and numerical values with excellent agreement.

Numerous homogenous premixed runs were performed and aggregated flame surface speed data from the high speed Schlieren setup was used to determine the flame front propagation speeds of the homogeneous premixed mixtures and values were compared to literature values. Measurement of dilute gas mixtures were examined using the gas analyzers and provided additional confirmation that the burned gas equivalence ratio was achieved. After each experimental run, gas samples were measured and this procedure provided an additional validation method to ensure a known tank equivalence ratio and provide a basis for comparison with the stratified experimental results.

In addition to extensive post-processing of the Schlieren images, to better quantify the soap bubble's affect on the flame propagation a set of experiments using a series of homogenous mixtures with the soap bubble were run.

3.4 Experimental Setup: Stratified Flame Propagation

To controllably form a stratification layer is quite a formidable challenge. Fundamentally, entropy fights experimentalists on developing a controlled and sharp interface between two miscible gaseous mixtures. As such, a variety of techniques to develop a stratification layer were brainstormed. Due to various equipment, financial, and time limitations, only a subset of the possible setups were tried but ultimately a soap bubble was chosen as the best option.

Evaluation of the Stratification Setup

Ideally a diagnostic tool such as planar laser-induced florescence (PLIF), chemiluminescence, gas chromatography (GC), or nondispersive infrared (NDIR), would be able to determine the spacial concentration profile of methane or its combustion. However, none of these tool were available for this work and therefore alternative methods were devised as a proxy to estimate the stratification layer. To reduce the uncertainty of the layer, extensive validation and procedures were undertaken to improve the separation boundary as best as possible.

As past experiments of gaseous methane permeating through pure water have indicated a low diffusion rate of about 1.88E-5 cm^2/s[61] compared with methane in air with a diffusion rate of about 0.2 cm^2/s[11], [38]. Therefore, it was initially assumed that soap mixtures that contain mostly water would exhibit the same desirable high permeability characteristics observed in the literature. Additionally, as prior work has not indicated permeability issues with using soap bubbles to separate mixtures, this assumption was deemed initially appropriate. However, rigorous testing of the permeability was deemed necessary following odd behavior in the stratified flame experiments.

Permeability through Bubble

Therefore, to test the efficacy of the bubble to separate two mixtures adequately, a number of systematic tests were devised. One key test used was found to have been previously used by Karim et. al [30] in which a sparking system is used to detect the ignitability of the flame and therefore evaluate the diffusive characteristics of the mixture. Such a method was modified to test the permeability of gas through the bubble with the main goal to determine the time at which the rich ignition limit would be reached when the soap bubble initially contained too high of fuel to ignite and air in the rest of the chamber. With such knowledge, an estimate of the combined effect of the permeability and diffusion of the mixture within the bubble down to the upper flammability limit could be made and an evaluation of the separation between the bubble and the outer mixture could be evaluated.

The test was setup by blowing a super rich, ϕ greater than the upper flammability limit ignitable in the setup, bubble and using air in the outer mixture within the chamber. A series of systematic sparking tests were then performed to determine if and when the mixture would be ignited, indicating sufficient permeation and molecular diffusion to support ignition. At different times following a successfully blown bubble, a spark was used to attempt to ignite the mixture within the bubble. If the mixture within the bubble didn't ignite, the mixture and time were noted and the chamber was vacuumed before a new bubble was blown and a different time to ignition was set.

Following these tests, it became clear that considerable care must be taken to minimize the permeability through the bubble and the subsequent diffusion that follows. Initially the mixture would not ignite. However, following a finite time that depended on the specific soap mixture, the mixture became ignitable. Later times again lead to a condition where the mixture would not ignite, indicating passing into the lean ignition limit.

It was noted that mixtures with more dilute surfactants in the soap mixture provided increased resistance to permeability, however at the cost of both the integrity of the bubble and additional influence of the bubble on the flame. Increasing the water content in the soap mixture reduced the surfactant count in the soap and eventually led to premature popping of the bubble prior to ignition. Increasing the water content also increased the surface tension of the water mixture, which holds the bubble in the spherical shape. With increased surface tension, the vigorousness of the bubble popping increased as appeared influence on the flame passing through the bubble induced disturbance region. A trade-off then became apparent between the desired popping characteristics and the desired low permeability through the bubble.

Stratification Setup Procedure

The mixture stratification layers were setup for a spherical flame geometry using a soap bubble to separate two homogenous premixed mixtures. First, the constant volume chamber was filled with the external gas mixture in the same manner as the homogenous premixed experiments. Then, a metal tube was dropped into a pan filled with a custom soap mixture. The tube was then vacuumed out with a vacuum pump and then filled with a second mixture. When the tube was lifted out of the soap-filled pan and set to a specified distance, a bubble of approximately 29 mm (1.14 in) in diameter was blown with a second mixture. The size of the bubble was controlled using a controlled volume and regulating the pressure as indicated by a gauge. The distance of the tube was set such that the bubble floated freely in the center of the optical area while at the same time the grounded bottom pin of the spark ignition system penetrated the bottom of the bubble. Figure 3.2 shows a schematic of the setup with a bubble within the constant volume chamber following the proceeding procedure has been followed.

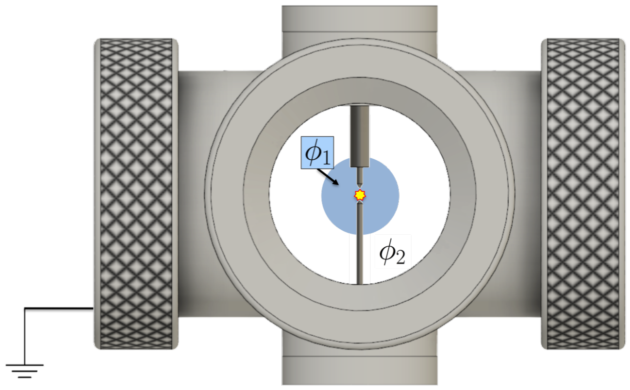


Figure 3.2: Stratified mixture experimental setup of side profile of constant volume chamber with soap bubble and enclosed mixture, ϕ_1 in blue and second mixture, ϕ_2, filling rest of chamber in white.

Following the bubble's formation, the top pin was descended though the metal tube and set a specified distance from the bottom pin. After a specified time from the initial bubble blowing, the inside mixture was ignited and the flame propagated spherically outward and, if necessary, burst the bubble to continue the propagation outward into the secondary chamber mixture.

Kelvin-Helmholtz Instabilities

As mentioned prior, much effort was focused on reducing the observed effect of the bubble on the flame structure. However, Kelvin-Helmholtz instabilities were limited yet unavoidable with the bubble stratification setup. When the bubble popped the bubble itself acted as a moving sheet of fluid relative to the stagnant surrounding mixtures, the characteristic of a velocity shear layer that induces Kelvin-Helmholtz instabilities. Similar to the two layer couette flow with an interface leading to interfacial stabilities as described by Charru and Hinch[6], the popping bubble generates vorticities within the stratified mixing layer. These vorticities enhance the mixing and with time reduce the sharpness of the stratification layer. Also, as pointed out by Govindarajan and Sahu[25], the origins and extent of miscible shearing layers has only recently been a focus of research. Subsequently, the three layer flow of two miscible mixtures separated by a third immiscible layer as the bubble pops from the leading edge transitioning into a disturbed two layer miscible flow is quite a formidable fluid dynamic challenge that is beyond the scope of this work. Fortunately, as demonstrated numerically by Geun and Kim[22], higher surface tensions, as in this work, leads to reduced growth of this instability. Therefore, this would indicate that the region of influence on the flame would be short lived in it's expanding propagation. From the Schlieren images, the flow around the bubble popping fluid was determined to be laminar, Re_D $1.6E4$ to $max(Re_{arc})$ $4.9E4$, with limited time influence on the flame. These instabilities appeared as ripples in the flame as the flame propagated through the flow instability region.

Emissions

Gas composition was determined by passing a test gas through a five-gas HORIBA Analyzer, measuring unburned hydrocarbons (THC), molecular oxygen (O_2), carbon monoxide (CO), carbon dioxide (CO_2), and nitrogen oxides (NO_x).

Exhaust gas of numerous experimental runs were collected and diluted for measurement with the gas analyzers in batches. To collect the products and evaluate emissions, an argon purged and vacuumed tank was connected to the constant volume chamber and filled with the exhaust gases after each flame experiment. After a set number of similar experiments were collected in the emission tank, the gaseous exhaust mixture was diluted with argon and allowed to mix for a specified time. The argon diluted exhaust gas was then measured using the gas analyzer, which was calibrated each day before use, and average values were recorded. Due to limitations with the equipment, CO measurements were unreliable and unable to properly calibrate, and so only estimates were recorded.

To further validate the homogeneous gas compositions, homogenous premixed unburned tank mixtures were diluted to match the capabilities of the gas analyzers and measured. The equivalence ratio was determined using the equation outlined by Brettschneider[5]. Good agreement was achieved in the values predicted by the partial pressure method, both for the unburned and burned emissions.

3.5 Experimental Setup: Flame Propagation through Electric Field

As this work was initially undertaken to understand the combined effects of stratification layers and electric fields, the experimental setup was modified to provide a quick and straight forward addition to the stratification layer setup. As numerous work has been done prior on the effects of electric fields, limited experimental work was performed with the electric field setup alone and focus was devoted to the stratification layer experiments following successful proof of concept design of the electric field setup. The designed electric field modified setup used for preliminary findings in this work for the electric field's effect is presented below and designed for use separately with homogeneous premixed mixtures as well as in conjunction with the stratified experimental setup.

Applied Electric Field Experimental Setup

The electric field setup was employed though the use of a thin wire mesh connected to the chamber by use of ceramic stand-offs and a high voltage cable connected to the meshes and a high voltage power supply provided the high voltage to the meshes. The chamber and the ignition pins were grounded following ignition. This has been similarly done by Meng, et. al [39]. A depiction of the mesh standoff arrangement used in this work is sketched in Figure 3.3.

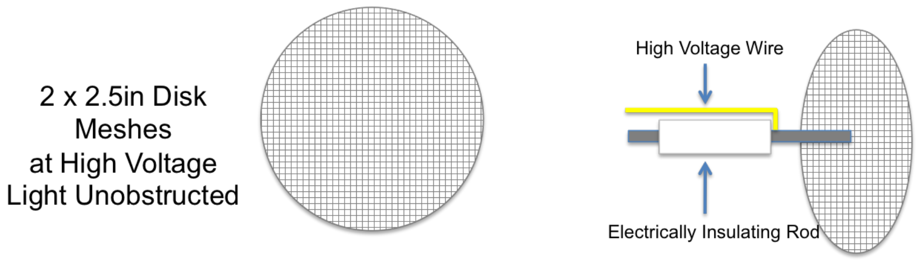


Figure 3.3: Schematic of mesh arrangement within chamber

The mesh plates were held at a specified voltage and powered by one of two model N030HA2/P030HA2 Acopian power supplies where were able to supply between 0 and ±30 kV, of which were connected with a switch to provide safe control and operation of the units. Voltage was set and monitored by a voltmeter attached to a reduced voltage readout. The flame was ignited and allowed to propagate after the electrostatic field stabilized and the resulting pressure and Schlieren images were captured. The voltage was limited to no more than ±9 kV as corona discharge and breakdown was noted at higher voltages. The electric field setup within the constant volume chamber is depicted in Figure 3.4.

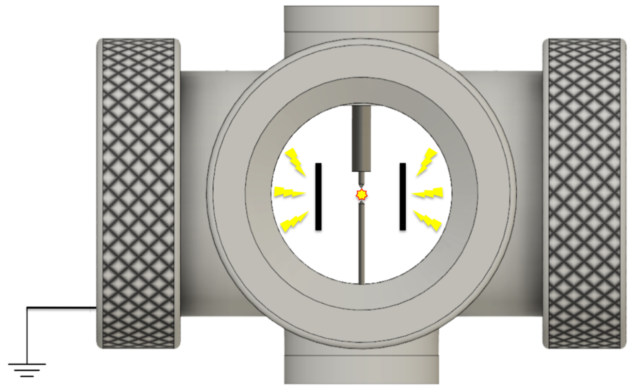


Figure 3.4: Constant Volume Chamber with optical access and enclosed mesh plates.

Chapter 4

Post-Processing: Determining Flame Speed

The raw image and sensor data captured by the high speed camera and data acquisition system for each run was output into avi container of uncompressed Schlieren image frames and text files that were saved to the hard drive for later processing.

A custom edge tracking post processing code was developed to accurately and repeatably interpret the flame speed data from Schlieren high speed camera frame data. The code was developed using the MATLAB language, but structured in a way to easily be adapted for use in Python or other similar interpreter.

Overall, the post-processing involved converting the image data to flame radius as a function of time. The unstretched flame surface speed was determined using the stretch rate calculation in Equation 2.19 and the non-linear zero stretch extrapolation calculation using Equation 2.21 to determine the unstretched flame surface speed. Finally, the laminar flame speed was determined once accounting for the expansion effects using Equation 2.22.

For the homogeneous cases, the laminar flame speed was also determined.

4.1 Image Post-Processing: Robust Interface Tracking of Dynamic Radial Surface Fronts

Limitations of Built-in Image Processing

The goal of the initial phase of the image processing was to determine the location of the flame at each frame.

Various image processing techniques and packages have been developed are available for straight forward use, and were tried, including Sobel, Canny, Prewitt operator, among others. Most of these methods rely on a thresholding technique to determine the interface between two regions and subsequently to determine the edge of a closed surface. These methods ideally will detect all of the possible edges in an image. However, in reality there are errors in the detection due to noise in the experimental data. In addition, difficulty arises in selecting the appropriate edge within the superset of edges detected. With the Schlieren, numerous edges are detected within and out of the flame of which thresholding alone was unable to separate. The strict built-in thresholding techniques partially succeeded in locating the flame front of homogeneous premixed flames, but more fine tuned control was necessary. Aberrations due to noise induced from the Schlieren setup, such as a 'dirty' window, gave off irregular lighting gradient effects that the built-in codes were unable to reliably differentiate from the flame surface. Furthermore, however successful the built-in packages were at determining the flame surface of a homogenous premixed flame, the methods struggled when the 'shadow' of a bubble was present. The tracking algorithm could not differentiate a bubble's surface with a flame's surface, and did not incorporate any history or physics to where the flame surface was physically likely to be. In addition, differentiating flames from bubbles was paramount and bubbles were present in a large portion of the experimental runs. In particular, since the concentration gradient was sharpest at the bubble interface for the stratified flame experiments improper measurement of the flame surface could lead to inaccurate conclusions when smoothing alone is used to account for the differences.

Unfortunately, due to the variability of the Schlieren images and the varying lighting effects imposed by using vertical knife edge Schlieren, a customized in-house code was needed to accurately and repeatably capture the flame front and, where applicable, simultaneously the bubble interface.

Flame Surface Tracking: Overall Algorithm Employed

A custom code was built that incorporated the physical limitations of the experiment and tuned to best handle the recorded data. Key image reference data was selected manually and checked by the code, but the surface tracking was developed to more robustly and autonomously track the flame and bubble surfaces. A modular approach was used to determine the location of the flame edge.

As most of the equivalence ratios considered had expected flame speeds greater than 15 cm/s and thereby limited buoyant effects, a single row was used initially to determine flame surface speed. However, after later development, this code was expanded to include a significantly broader set of pixel rows to determine the location of the flame even more accurately and dynamically, where the surface front was temporarily not smooth as in many of the stratified cases.

For each row, the code process followed a similar methodology to the Canny edge detector scheme as follows:

- Determine region where flame physically possible
- Within this region, smooth the brightness data and spline fit for additional points
- Given desired estimated thresholding criteria, determine possible locations of the flame and location of a bubble

- Evaluate and select the flame location from the set of possible points

To determine the region to search for the flame, flame history was incorporated along with an expected upper bound to the flame speed where a flame was physically possible to be found. For the cases where a bubble was possibly present, a similar method to determine the physical search region was used, incorporating the maximum lifespan of the bubble.

Then, within the physically possible flame location area, a target brightness threshold was searched for to determine the set of points that may indicate the location of the flame. Then, each row's brightness values were smoothed to remove irrelevant sensor noise and aberrations due to the experimental setup. A spline fit was used to interpolate between the pixels to better locate the flame front. In homogenous experimental cases, the target brightness corresponded to the average temperature between the flame and the unburned mixture. For the experiments where a bubble was present, the tracking brightness chosen corresponded to a significantly higher temperature as the fluctuations in the density of the flame and bubble prevented the average temperature from being selected. This was deemed acceptable in these cases as the flame thickness was thin and on the order of 1 mm. The selected brightness targets were chosen but tolerance was allowed to account for fluctuations in the flame as well as to account for when the bubble popped.

For each possible flame location found, a series of tests were performed to located the true location of the flame and differentiate it from other causes. The slope of the brightness data was also tested to ensure it aligned with the expected positive density gradient of an expanding flame. In addition, each of the found thresholded targets were tested for other possible cases, such as a bubble, and differentiated so as to select the best location of the flame.

From each row processed, the number of which depending on the size of the flame, the location of the flame found was aggregated to determine the leading flame surface front location. Multiple rows allowed for flexibility and error mitigation in finding the flame location. In addition, as the flame was not smooth during many of the stratified cases, unlike the homogenous cases, aggregations provided a better way to estimate the overall surface speed.

A visual example of the code tracking a flame and bubble can be seen in Figure 4.1. The flame is tracked in red, while the bubble is tracked in blue. Excellent repeatable and robust tracking was achieved with the code and the radius data was used for later additional post processing.

Flame Surface Tracking: Geometric Considerations

To account for the flame stretch and evaluate buoyant and ripple effects, various geometric properties were calculated. Initial geometric considerations were determined, given the pin-pin ignition center as the center of the spherical flame. The radius of the flame was determined as the difference between the pin-pin ignition center and the geometric distance to the flame surface at the leading edge. However, in some cases, such as very slow homo-

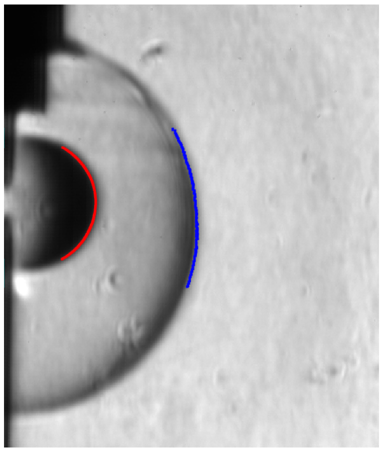


Figure 4.1: Example of post processing of flame tracking, in red, and bubble tracking, blue.

geneous premixed flames, the single row flame tracking algorithm inaccurately located the flame as buoyant forces drove the flame upwards. In addition, there were some cases where the bubble moved the flame as it popped or the flame propagated slowly and was subjected to buoyant driven movement upwards. These and other cases necessitated an increase in the robustness of the flame tracking algorithm and to better account for the radius of curvature needed to determine the flame stretch each of the flames were subject to.

To determine the spherical radius necessary for alternative determination of the flame stretch as well as track the flame leading edge for flames that were subject to buoyancy, multiple row tracking was implemented and the information from each row location were aggregated to estimate relevant geometric properties. As spherical flames have a circular projection the best fit circle through all of the flame surface tracked pixels was sought.

The flame surface location of multiple rows at each frame were grouped together into a set to calculate the best fit circle through the set of points detected. In addition, the location of the bubble's center and radius were determined from a best fit circle from a series of detected points along the bubble. The best fit circles were determined using an algorithm outlined by Gander et. al[21], where the minimization problem is solved for the distance between the circle's ring and the distance from a set of data points. In particular, it involves solving the following minimization problem for a set of m data points located at (x_i, y_i) by selecting the best circle, (x_{cen}, y_{cen}, r) that satisfies equation 4.1.

$$\sum_{i=1}^{m}(||(x_{cen}, y_{cen}) - (x_i, y_i)|| - r)^2 = min \tag{4.1}$$

This minimization was solved using an iterative method that used a first guess to minimize the algebraic distance of the circle and the points of interest. The successful minimization gave the circle center and radius of the best fit circle and closely inscribed the points of the rows found of the flame locations. From the best fit circle, the apparent circle center was used to compare both an additional, apparent, flame radius for each row flame location as well as identify the row location of the leading flame front. A depiction of a typical set of detected points from the flame surface with the flame tracking points and the center of these points in red can be seen in Figure 4.2.

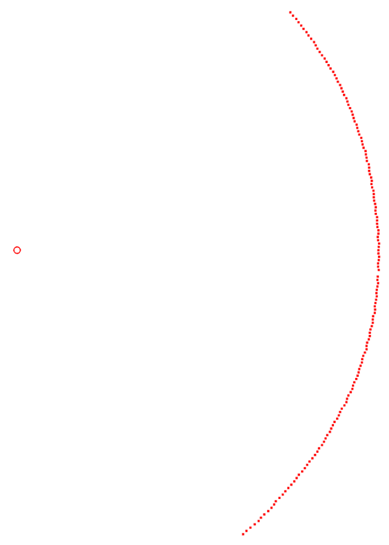


Figure 4.2: Typical flame tracked rows as red dots with flame best fit circle center as red circle.

The leading flame front location was determined based on the y-location of the circle center. This allowed a close monitoring of buoyant effects on the overall movement of the flame.

If the flame is assumed a perfect sphere, the flame will follow a circular shape in the Schlieren image. However, due to heat transfer to the pins and straw as well as the initial shape of the ignition regime, the flame initially develops a slight toroid in shape, as confirmed by simulation. This slight deviation could then be evaluated to compare with the spherical assumption.

Flame Surface Tracking: Influence of the Bubble

As previously mentioned, the cases where a bubble was also present in the chamber was a contributing factor in generating the custom code. As mentioned in the literature, the first frame can often be subtracted from the rest of the frames and the difference in the brightness is used to track the flame surface. However, in many cases considered the background was a large function of time, complicating the strict image subtraction procedure. In the cases where a bubble was present, the bubble acts as an additional moving surface boundary that is present from the first frame and which expands and eventually pops, disrupting the flame's background image. Therefore, the usual frame subtraction technique was not possible and the bubble location was instead tracked in a similar way as the flame to limit the impact.

Furthermore, as the flame approached the bubble, in some cases the differentiation between the bubble and the flame became nearly impossible to distinguish. However, the flame's location could be deduced with reasonable accuracy over this short zone and time span.

For some mixtures far away from stoichiometric values, the flame was significantly slower the flame moved upwards due to buoyancy. Additionally, for these same mixtures when flames propagated through a bubble, the bubble at times pushed the flame upwards significantly. In both of these cases, the leading edge of the flame was tracked using the using the row of the apparent circle center after solving Equation 4.1.

Code Speed Improvements

The image processing to determine the flame radius as a function of time took considerable computational effort. Initially with the manual method of radius gathering a video could be processed somewhere between 3 and 10 minutes, depending on the length of the video. Speed improvements were initially gained with the initial automatic method of a single row point used, the processing time significantly decreased with less human interaction required. However, tracking additional rows as required by geometric determinations increased and at times orders of magnitude increases in the time to processes a video. This prompted improvements to the speed of the image processing code to efficiently work with the experimental data.

Initially, the workflow was augmented to allow for a decoupling of the image processing and subsequent post processing. However, it soon became clear that this alone was not adequate. As changes were made to the code to improve the capabilities and robustness of the code, continued profiling highlighted relevant gains. Various serial and parallel programming techniques were then employed successively to incrementally speed up the image processing code. Speed-ups of multiple orders of magnitude compared to the initial multi-row code development were performed through astute profiling and subsequent code enhancements. Some of these process speed enhancement methods utilized in the code development included improved memory management and access time, parallel processing, and grouping of similar tasks.

Specific focus was given to memory management where the data storage within the post-processing code was restructured to reduce access time. The hard drive input/output calls and operations were limited where possible, and reads and writes were grouped together. Other focus was given to parallel running techniques where multiple subprocesses were run in parallel. This iterative speed-up processed allowed for additional image post-processing to be done in a reasonable amount of time as the number of experimental videos quickly grew.

4.2 Post Processing: Radial Flame Surface Speed

The raw radius data gathered from the image edge detection algorithm was interpreted to interpolate and determine the flame speed. The algorithm was structured as follows:

- Cropping endpoints of raw radius data for relevance
- Filtering of radius data
- Spline fit interpolation of radius data
- Calculate the flame surface front displacement speed
- Determine the unburned gas expansion factor
- Calculate the stretch rates the flame experiences
- Evaluate flame speed

The raw radius data was cropped to only include the time from when the flame was ignited until the flame reached the edge of the optical frame. To minimize ignition effects, in this work a radius of greater than about 1 cm was deemed no-longer subject to ignition effects, while lower values have been used in the literature. The selected data was then lightly filtered using a moving least squares fitting algorithm. Care was taken to minimize the diffusive characteristics of the filter, and parameters were chosen judiciously. The third order Savitzky-Golay filter with a frame length of 15 for homogenous flames without a bubble, and a frame length of 9 was chosen for cases with a bubble. These values provided a good balance between reducing noise without removing or changing relevant features of the flame surface.

The radius data was interpolated using a spline fit before the flame surface front displacement speed was calculated using a fourth order 5-point centered finite difference first derivative approximation from the radius data as given in equation 4.2.

$$\left(\frac{dr}{dt}\right)^j = \frac{-r^{j+2} + 8r^{j+1} - 8r^{j-1} + r^{j-2}}{3(t^{j+2} - t^{j-2})} \tag{4.2}$$

The stretch rate was then determined from the flame surface speed and the radius. The flame surface speed vs smoothed stretch rate was then used to extrapolate to an unstretched flame speed using a fit of the data from a radius of about 1 cm to 2.5 cm to eliminate the ignition effects. Both a linear fit as given in Equation 2.20 and a nonlinear fit as given in Equation 2.21 were computed and used in the zero stretch extrapolation The nonlinear method was used in the results as this method is noted to be superior in the rich equivalence ratio region, as given by [9] et. al.

To determine the laminar flame speed of the homogenous mixtures from the unstretched flame surface speed, the expansion of unburnt gas was used to estimate the expansion effects.

The initial gas temperature from the DAQ was used as the unburned gas temperature while the burned gas temperature was estimated as the equilibrium temperature of the specific equivalence ratio as calculated from the CHEMKIN code EQUIL.

4.3 Validation of Code

Validation of the flame tracking post-processing code was performed first by comparison to the initial, manual human input edge tracking program as well as comparison to previous literature values. First, the frames were read in and the same user inputs as the automatic flame detection system were used. Then the location of the centerline of the flame was selected for each frame to build up the radius data manually. The radius was then compared with the automatic flame tracking algorithm, and excellent agreement was achieved with minimal deviations.

Additionally, the homogenous data was processed, and the unstretched laminar flame speed was in good agreement with past literature values and within literature uncertainty.

The relevant post processing codes developed for this work can be found in Appendix A.2.

Chapter 5

Numerical Simulation and Modeling

5.1 CHEMKIN Collection

The CHEMKIN Collection[43] is a library and set of application tools initially developed at Sandia National Labs to determine and analyze gas phase chemical kinetics. The libraries incorporate and evaluate thermodynamic properties, transport properties, and gas phase chemistry from a chemical kinetic mechanism. The base subroutines evaluate gas-phase multicomponent viscosities, thermal conductivities, diffusion coefficients, and thermal diffusion coefficients. These calculated properties and data are used to determine chemical kinetics relevant for combustion and coupling with fluid dynamic solvers. There are a number of modules developed and adapted from the base library, and a few of them are referenced and used in this work.

Chemical Kinetic Mechanism

The library requires a chemical kinetic mechanism which details the mathematical model of the complex chemistry reactions. The layout follows a specific format including the following: species, reactions, and relevant parameters for specific model used. Various chemical reaction models can be used including an Arrhenius rate model, the Lindemann formulation, the Troe formulation, the SRI formulation, and the Landau-Teller formulation. In addition, duplicate and reverse reaction rates can be specified. For this work, the reduced mechanisms used were GRI 1.2 and GRI 3.0[55].

PREMIX

PREMIX is a semi one dimensional planar isobaric unstretched premixed laminar flame speed solver developed as part of the CHEMKIN collection[43]. It solves the boundary value problem needed for predicting the laminar flame speed given temperature, pressure, gas concentration, gas transport properties, and a chemical kinetic mechanism. The PREMIX code converges to the steady-state premixed-flame solution on a finite-difference discretization, simultaneously solving the continuity, energy, species and equation of state. More specifically, PREMIX solves the continuity equation, Equation 5.1, the energy equation for K species, Equation 5.2, and the species equation, Equation 5.3, and the ideal gas equation of state.

$$\dot{M} = \rho u A, \tag{5.1}$$

$$\dot{M}\frac{dT}{dx} - \frac{1}{c_p}\frac{d}{dx}\left(\lambda A \frac{T}{dx}\right) + \frac{A}{C_p}\sum_{k=1}^{K}\rho Y_k V_k c_{pk}\frac{dT}{dx} + \frac{A}{c_p}\sum_{k=1}^{K}\dot{\omega}_k h_k W_k = 0, \tag{5.2}$$

$$\dot{M}\frac{dY_k}{dx} + \frac{d}{dx}\left(\rho A Y_k V_k\right) - A\dot{\omega}_k W_k = 0 \tag{5.3}$$

For this work, multicomponent transport properties were calculated using the CHEMKIN subroutines and the premixed combustion flame computations were performed to determine the adiabatic freely propagating flame speeds for comparison to the laminar flame speeds determined in the experiment and in literature. To do so, PREMIX solves the eigenvalue problem for the mass flow eigenvalue $\dot{M}$ by simulating the flame in the Lagrangian reference frame, reducing a degree of freedom. The resulting adiabatic laminar flame speed gives an upper bound of the expected experimental laminar flame speed.

PREMIX was used in this work to estimate the laminar flame speed for comparison to experimental and other numerical methods. In addition, PREMIX was used to estimate the influence of the soap solution and water vapor within the chamber for varying equivalence ratios. More specifically, the adiabatic flame speed was determined with a worst case saturated water condition to evaluate the effect of water, the main component of the soap, on the flame speed for each of the concentrations. This provided a comparative flame speed reduction when introducing water vapor to the flame.

EQUL

EQUL is also part of the CHEMKIN collection[43], which calculates the equilibrium concentration at steady state through the use of STANJAN library subroutines. This FORTRAN code does this by minimizing the total mixture Gibbs free energy. The code evaluates the Gibbs function, $g_k(T,P)$, for K species at the temperature and pressure and constrained to conserve atomic species. Mathematically, the minimization problem is expressed in Equation 5.4 for the sum of N_k moles of each species k.

$$\sum_{k=1}^{K}\left(g_k(T,P) + RTln\left(\frac{N_k}{\sum_{k=1}^{K} N_k}\right)\right) N_k = min \tag{5.4}$$

In this work, EQUIL was used to determine the equilibrium concentrations and temperature at constant pressure and enthalpy problem to account for the compensation needed for the unburned gas density expansion effects of the initially expanding flame not accounted for in the extrapolated zero-stretched flame speed, as proposed by Giannakopoulos[23].

5.2 One dimensional Compressible Flow Solver

A one dimensional radial unsteady reactive flow code with adaptive mesh refinement, named Adaptive Simulation of Unsteady Reactive Flow (ASURF), was compared with the experimental spherical flames building off past work by Shi[54] on spherically expanding methane-air flames.

This code solves the one dimensional radial unsteady mass, momentum, species, and energy conservation equations along with the ideal gas equation of state. ASURF uses the CHEMKIN subroutines to calculate the transport properties needed for the multi-component transport from Fourier's, the mixture-averaged diffusion velocity formula expanded from Fick's law, and Stokes' law. More details of the serial version of this code are given by Chen[10],[7]. A chemical kinetic mechanism is used with CHEMKIN to calculate the reaction rates at each time step.

As part of this work, the co-development of a MPI parallel version of this spherical flame solver, called ASURF-Parallel, was completed to speed up the time consuming computations. This included domain decomposition and splitting of the time dependent ordinary differential equation to distribute the computational work load across multiple processes. More details of this parallelization have been given by Shi[54].

This code assisted with determining initial design criteria to allow for optical visualization of the flame stratified layer transition. Furthermore, this work included comparisons with the simulated values of this one dimensional code with both homogenous and stratified flame experiments.

5.3 Three Dimensional Compressible Flow Solver with Chemistry

The commercial three dimensional solver CONVERGE Version 2.3[47][46] was used to compare with the experimental results. The compressible fluid dynamic equations were computed for a similar geometry as the experimental work using a finite volume method with adaptive mesh refinement (AMR). Namely the mass conservation equation, Equation 5.5, the momentum conservation equation, Equation 5.6 , for each species k, the species conservation equation, Equation 5.7, and the energy conservation equation, Equation 5.8, along with the Redlich-Kwong equation of state were numerically computed. The orthogonal, structured grid for these simulations were automatically generated at runtime using a modified cut-cell Cartesian grid generation method from a set of surface geometry files derived from the experimental setup.

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = \sigma, \tag{5.5}$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_j} = -\frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2\mu\delta_{ij}}{3}\frac{\partial u_k}{\partial x_k}\right) + \sigma_i \tag{5.6}$$

$$\frac{\partial \rho_k}{\partial t} + \frac{\partial \rho_k u_j}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\rho D_t \frac{\partial Y_k}{\partial x_j}\right) + \sigma_k \tag{5.7}$$

$$\begin{aligned}\frac{\partial \rho e}{\partial t} + \frac{\partial u_j \rho e}{\partial x_j} = -P\frac{\partial u_j}{\partial x_j} + \left(\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2\mu\delta_{ij}}{3}\frac{\partial u_k}{\partial x_k}\right)\frac{\partial u_i}{\partial x_j} + \frac{\partial}{\partial x_j}\left(K\frac{\partial T}{\partial x_j}\right) \\ + \frac{\partial}{\partial x_j}\left(\rho D \sum_m^M h_m \frac{\partial Y_m}{\partial x_j}\right) + \sigma\end{aligned} \tag{5.8}$$

Two one-quarter geometries of the constant volume chamber were used to explore the three dimensional effects. One case included the pin center to evaluate the effects of the pin heat transfer on the shape of the flame. In addition, the same geometry but without the pins was used to model the rest of the three dimensional effects, and is reference in Figure 5.1. With the use of adaptive mesh refinement (AMR) to apply the grid spacing where necessary, the finest grid spacing used in this numerical work for the one quarter constant volume chamber geometry was 1.25 E-1 mm.

All of the mixtures within the three dimensional models were initially set to 300 K and 1 atm. The bubble was modeled as an ideal separation between the two mixtures, and was achieved through the use of a disconnect region that is disabled upon start of the simulation. The outer faces of the constant volume chamber were modeled as constant temperature Dirichlet boundaries set at 300 K. The internal pseudo faces, where the constant volume chamber was quartered due to symmetry, were set to a symmetric boundary condition, thereby adiabatic along the x and y faces fully shown in Figure 5.1. Ignition was performed by depositing 300 mJ over a 3 mm radius sphere over 2 ms. Chemistry was modeled using a reduced 22 species mechanism, GRI 1.2.

For the flame surface structure calculations, the maximum grid count was unconfined to allow the AMR to resolve the important features as required. For evaluation of the confinement effects, a more restricted maximum was used to limit the computational cost and memory required to execute the simulation.

Specifically in this work, the three dimensional numerical modeling was performed to estimate the confinement and heat transfer effects on homogeneous and stratified flames due to the experimental setup and conditions. Results were use to verify the flame shape and explore various three dimensional effects.

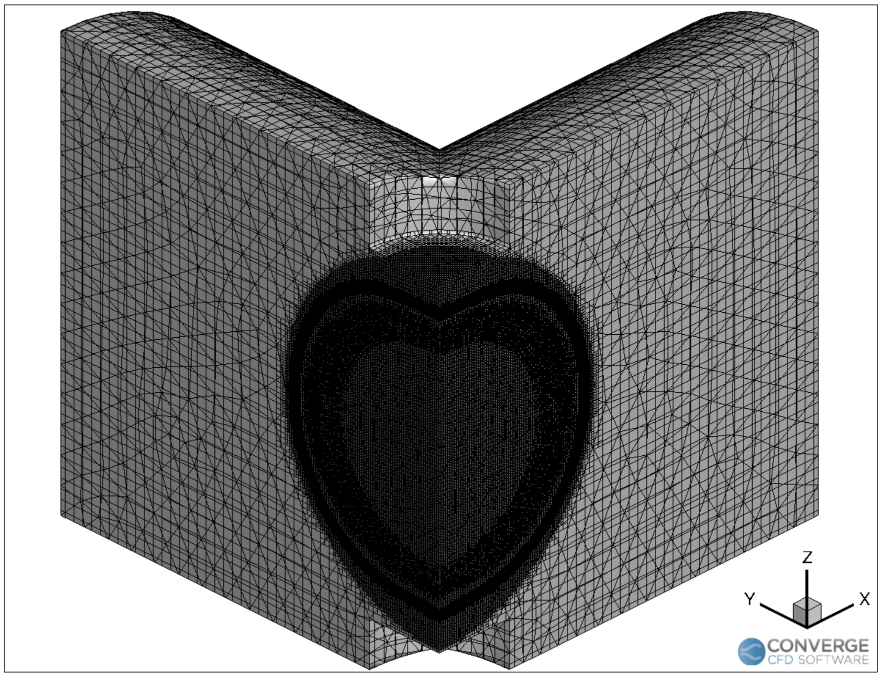


Figure 5.1: Quarter constant volume chamber geometry with example refined mesh through AMR, as used in this work.

Chapter 6

Experimental and Numerical Results

Experimental and numerical homogeneous and stratified flames were explored. As extrapolation to the laminar flame speed is not well developed for the stratified cases, the flame surface speed is compared in subsequent results and discussion.

6.1 Experimental Findings

Nearly spherical flames were experimentally studied using premixed methane-air mixtures of rich, lean, and near stoichiometric in one of several configurations.

Homogeneous premixed mixtures were tested to ensure internal consistency and agreement with literature values of laminar flame speed. These mixtures were also used to evaluate the influence of the bubble by testing a flame propagating from one mixture within the bubble to another mixture in the rest of the chamber.

Homogenous Spherical Flames

Laminar Flame Speed

Validation of the experimental setup and further additional homogenous runs were performed with good agreement with the literature. Laminar flame speeds in this work for near ambient conditions (105 kPa, 15°C) of various mixtures used were determine from tracked displacement speed that was extrapolated to zero stretch using the nonlinear fit method given in Equation 2.21, corrected for expansion effects, and compared with the literature. Good agreement was achieved and a summary of the experimental homogeneous tanks compared with literature values is presented in Figure 6.1. Deviations in the laminar flame speed for each of the mixtures is in relative agreement with literature values and satisfactory mixture composition was noted.

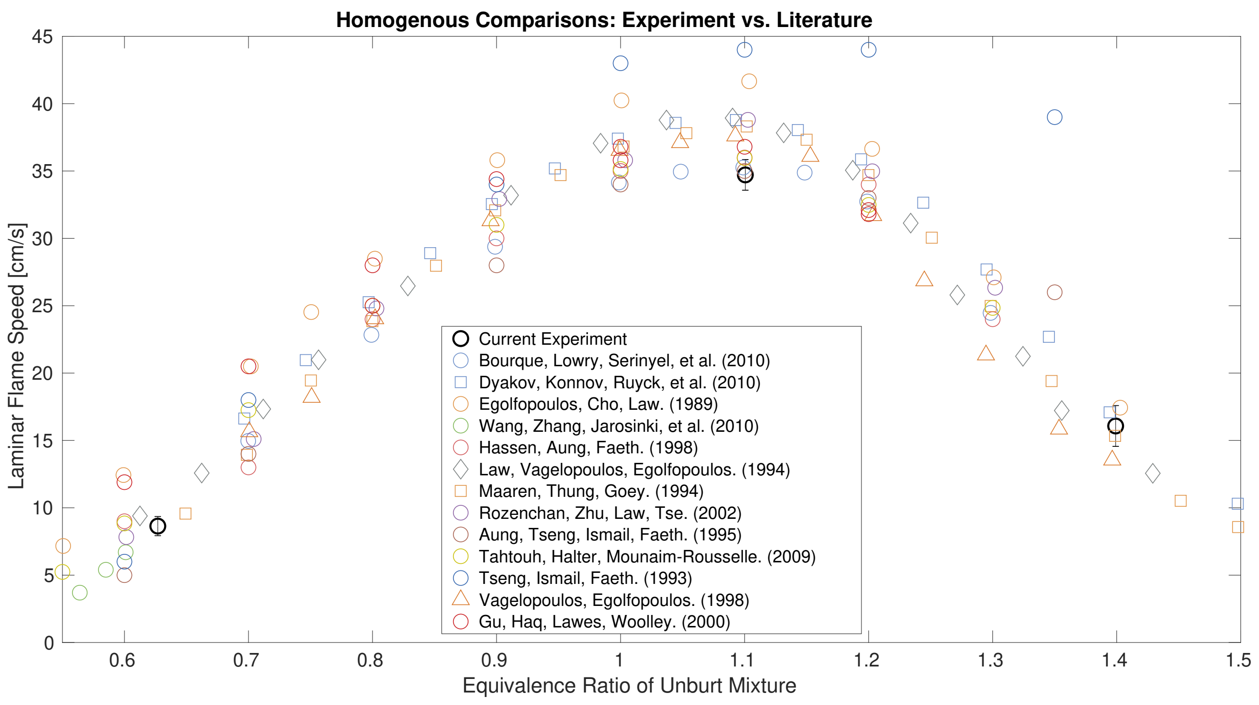


Figure 6.1: Laminar flame speed of homogenous mixtures compared to prior literature values, where similar spherical flames measurements are plotted as circles.

Emissions

Both unburned gas and product exhaust gas were measured in a purged collection tank. Exhaust gas emissions were measured for comparative analysis as well as used to validate the equivalence ratio.

Exhaust gas was collected for relative comparison of different mixtures. The tank mixtures within each of the mixing tanks were measured both as made and after combustion. Resulting emission measurements were used to determine the equivalence ratio, using the method outlined by Brettschneider[5]. The resulting calculated equivalence ratio of the tank mixtures agreed well with the expected equivalence ratio from the partial pressure method.

Additionally, measurements of product gas emissions for the stratified cases are presented in Tables 6.1 and 6.2. For comparison, emission data for a set of stratified runs of rich ($\phi = 1.4$) to lean ($\phi = 0.63$) are compared with a set of runs of an equivalent homogeneous mixture composed of the exact two stratified components without a bubble and injected into the constant volume chamber in the same quantity injected with the bubble and allowed to mixed for 2 minuets, in Table 6.1. The stratified rich ($\phi = 1.4$) to severely lean ($\phi = 0.5$) mixtures are presented in Table 6.2.

Run Type	THC [PPM]	O_2 [%]	CO [PPM]	CO_2 [%]	NO [PPM]
Rich $\rightarrow$ Lean Stratified	1982.66	4.88	146.33	5.08	15.75
Homogeneous Equivalent	2,421.37	5.22	114.98	5.35	36.99

Table 6.1: Comparison of stratified rich to lean flame emissions verses lean equivalent homogeneous emissions.

Run Type	THC [PPM]	O_2 [%]	CO [PPM]	CO_2 [%]	NO [PPM]
Rich $\rightarrow$ Too Lean Stratified	36,304.60	12.97	34.70	0.33	0.48

Table 6.2: Stratified rich to too lean flame emissions.

Homogenous Spherical Flames Propagating through Bubble

Besides strictly homogenous flame propagation, the influence of the soap bubble was rigorously tested. Various tests were performed to evaluate the effect of the bubble by filling the bubble with the same tank mixture as filling the constant volume chamber with. Results are presented below, and the makeup of the soap solution eventually used in this work is presented in Table 6.3, which was combined in lab glassware and covered with Parafilm before using the following day. The soap base solution used in this work goes by trade name *Natural Castile Liquid Soap Base* and distributed by Bramble Berry. The soap base consists of a blend of water, Olea Oil, Glycerin, and Potassium Hydroxide.

	Distilled Water	Soap Base	Glycerine	Total
Soap Solution:	96.4087%	3.5582%	0.0331%	100%

Table 6.3: Mass percentage of soap solution selected and used in this work.

Rich Premixed with Bubble

For flames propagating through homogeneous mixtures of premixed rich mixtures through a bubble and into the same rich premixed mixture, the surface tension of the popping bubble appeared to assist the buoyant upward movement of the slow flame. As the flame maintained a relatively spherical shape as it propagated and was pushed upwards, the leading edge of these flames was successfully tracked by the use of the row of the apparent circle center following determination of the apparent circle center from Equation 4.1. Fortunately, the bubble popping and subsequent propagation had minimal impact on the flame surface speed compared to the same rich mixture without a bubble.

Despite the strong influence on the location of the leading flame front edge, the structure of the flame surface was not impacted by the popping of the bubble and the relatively smooth surface of the flame can be seen. A typical case of a homogenous premixed rich mixture propagating through a bubble and into the same mixture, highlighting the effect of the bubble is presented in Figure 6.2. As can be seen in Figure 6.2, typical flames propagating

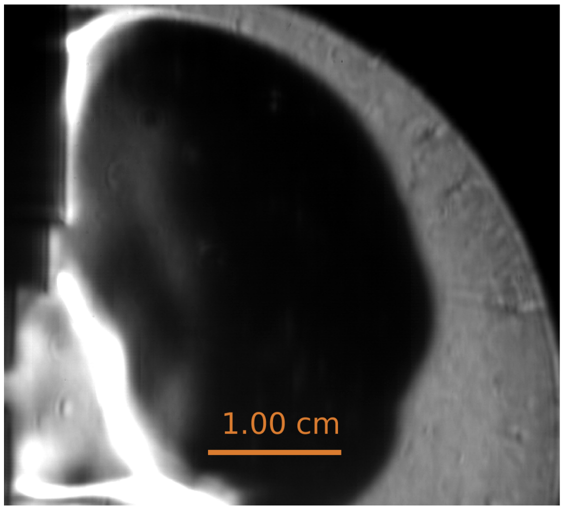


Figure 6.2: Typical Schlieren image of both the location and flame surface structure of the flame propagating from premixed rich mixture through the bubble and into same premixed rich mixture after propagating 3.18 cm from the ignition point.

from rich bubble mixtures into rich chamber mixtures after passing through the bubble layer did not appear to exhibit flame instabilities. The surface remained relatively smooth through out the flame propagation area, including the area where the bubble induced fluid dynamic instabilities.

Lean Premixed with Bubble

Similar to the homogenous rich flame cases where a flame propagated through a bubble's region of influence, the lean homogeneous bubble case had a similar set of limited influence from the bubble. The popping bubble also assisted with the buoyant movement of the slow flame upwards. Following successful tracking of the leading edge of having traveled 3.18 cm from the ignition pin center, the bubble exhibited little influence on the flame surface as can be seen in Figure 6.3. As can be seen in this figure and similar to the rich case, the structure of the flame surface was minimally impacted. For vigorous popping bubbles, some instabilities were seen in the flame for the lean to lean case. The structures of these flame surface deviations manifested as ripples and lines in the Schlieren image, normal to the direction of bubble popping.

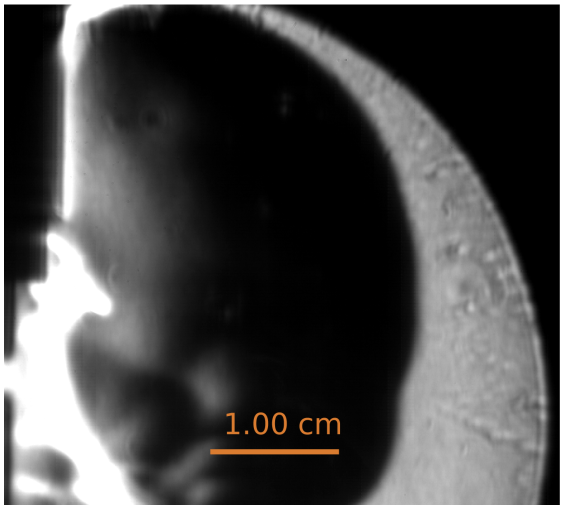


Figure 6.3: Typical Schlieren image of both the location and flame surface structure of the flame propagating from premixed lean mixture through the bubble and into same premixed lean mixture after propagating 3.18 cm from the ignition point.

Stoichiometric Premixed with Bubble

Unlike the rich and lean cases, the stoichiometric homogeneous mixture case with a bubble had slightly more influence from the bubble as the flame propagated significantly faster, leading to a much more vigorous bubble pop. The Schlieren images of a typical flame propagating from a stoichiometric mixture within the bubble to a stoichiometric mixture in the chamber can be seen in Figure 6.4. With the increased vigor of the bubble pop, the flame was pushed upwards and more instabilities were noted both in the fluid and in the flame structure. The flame surface exhibited some cellular features that slightly enhance the tracked flame surface speed.

Stratified Flames: Rich into Lean Flames

Given the relative limited influence of the bubble on the flame speed and flame surface structure of the particular experimental setup from flames propagating from a premixed mixture into the same premixed chamber mixture, stratified mixtures of rich within the bubble and lean in the chamber were prepared and flames were ignited and propagated.

The stratified cases were noted to exhibit a subdued bubble popping similar to both the bubble cases with either rich and lean mixtures. While the vigorousness of the popping bubble for the stratified cases was limited, the flame surface was noticeably changed as the flame approached the stratification layer. Similar to Figure 6.5, a significant cellular structure developed following propagation through the stratification layer. Interestingly, the cellular structure subsided as the flame continued into the lean mixture. In addition, the

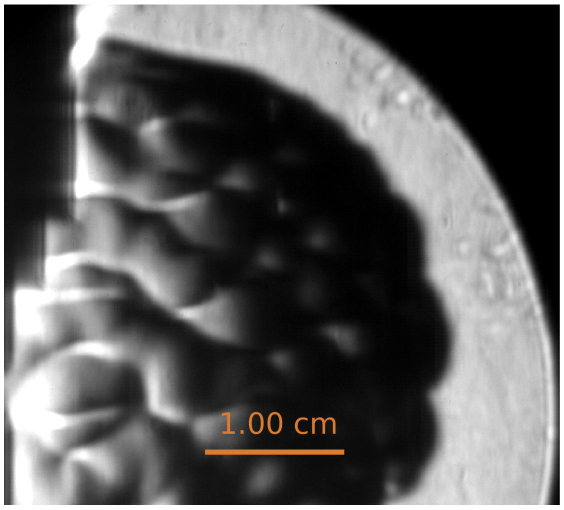


Figure 6.4: Typical Schlieren image of both the location and flame surface structure of the flame propagating from premixed stoichiometric mixture through the bubble and into same premixed stoichiometric mixture after propagating 3.18 cm from the ignition point.

tracked flame surface speed was noticeably higher during the transition region. A typical Schlieren image of the stratified flame propagation can be seen in Figure 6.5.

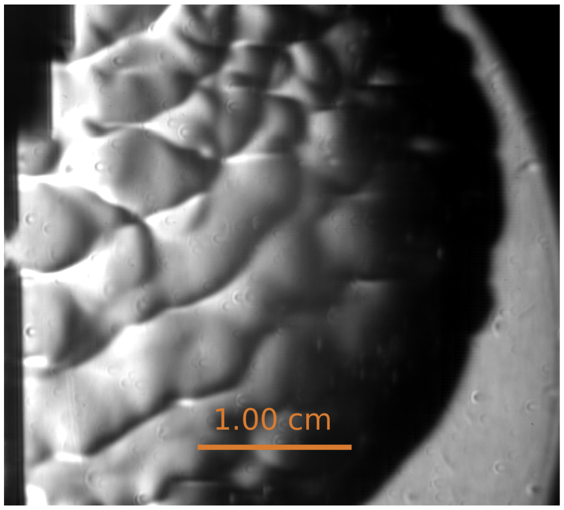


Figure 6.5: Typical Schlieren image of both the location and flame surface structure of the flame propagating from premixed rich mixture ($\phi = 1.4$) through the bubble and into premixed lean mixture ($\phi = 0.63$) after propagating 3.18 cm from the ignition point.

In addition to the cellularity seen in the flame structure, significant speed ups were noted in the flame surface speed. As has been done by others, the flame surface speed was calculated assuming that the speed up effects were only due to the stratification, including ignoring the flame surface structure effects and assuming it stayed spherical. Figure 6.6 shows a typical flame surface speed profile of a stratified flame propagating from a rich mixture and into a lean mixture, while Figure 6.7 shows an aggregated set of similar setup runs. The first two dips and first rise were attributed to the difficulty in tracking the flame surface as the flame approached the bubble. However, the second rise was noted to occur through the stratification layer and no difficulty was found in tracking the surface in this region. This flame surface speed is compared with a maximum flame speed for methane of equivalence ratio 1.1 and an equivalent homogeneous mixture composed of the exact two stratified components without a bubble and injected into the constant volume chamber and allowed to mixed for 2 minuets.

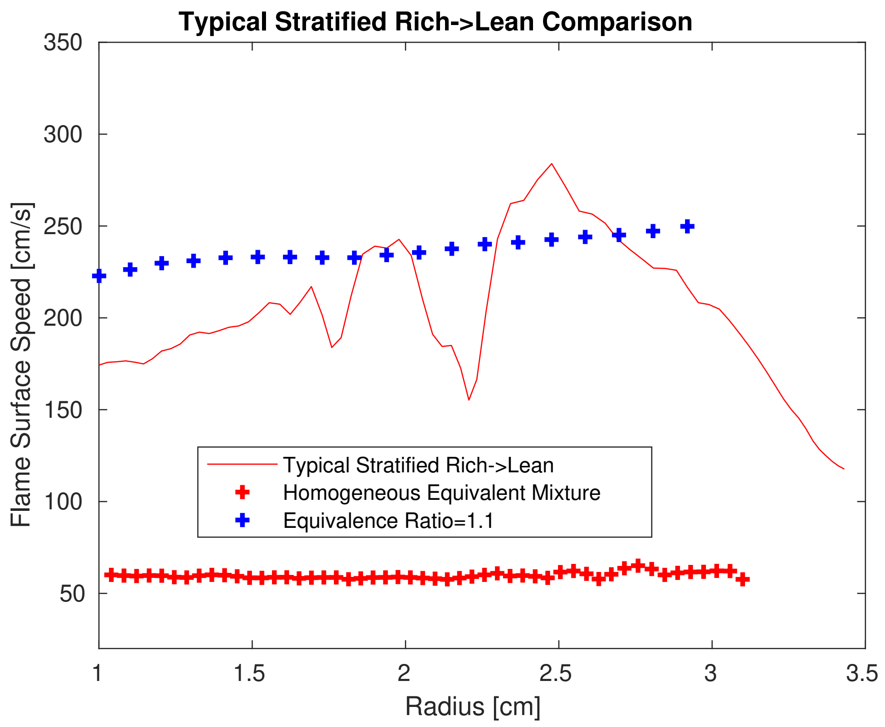


Figure 6.6: Typical flame surface speed of stratified rich ($\phi = 1.4$) to lean ($\phi = 0.63$) compared with equivalence ratio 1.1 and an equivalent homogeneous mixture of the two stratified components. The first two dips and first rise near 2 cm attributed to the difficulty tracking flame as it approached the bubble.

Interestingly, a similar flame surface structure effect as seen in Figure 6.5 was seen in the reverse case where a flame is propagating from a lean to a rich outer mixture but without the same enhancement of the flame surface speed. Figure 6.8 shows the lean ($\phi = 0.65$) to rich ($\phi = 1.4$) case also contained cellularities in all of the runs.

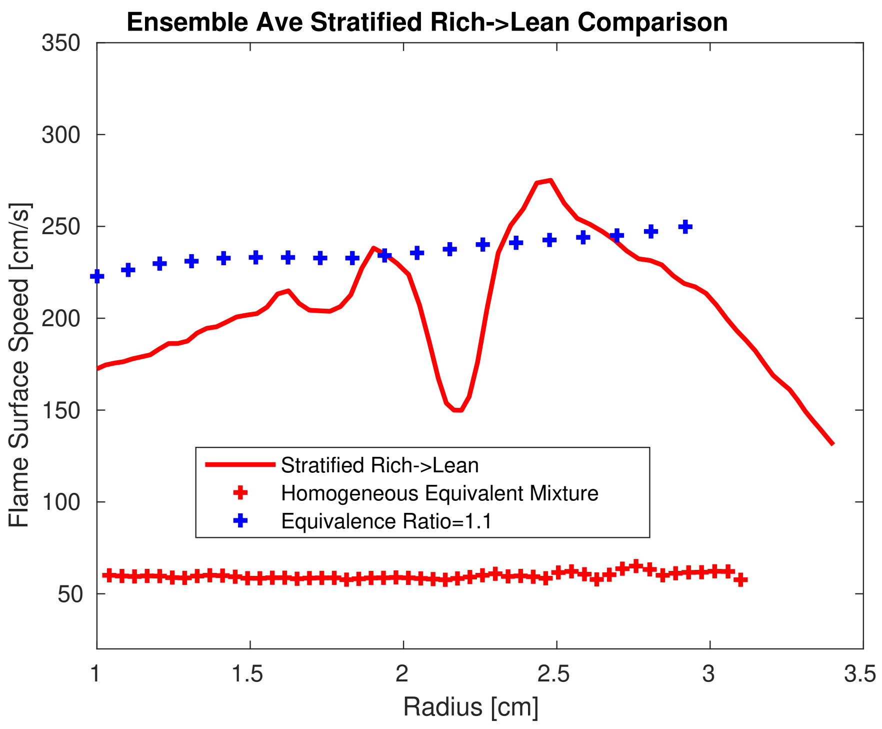


Figure 6.7: Aggregated flame surface speed of stratified rich ($\phi = 1.4$) to lean ($\phi = 0.63$) compared with equivalence ratio 1.1 and an equivalent homogeneous mixture of the two stratified components. The first two dips and first rise near 2 cm attributed to the difficulty tracking flame as it approached the bubble.

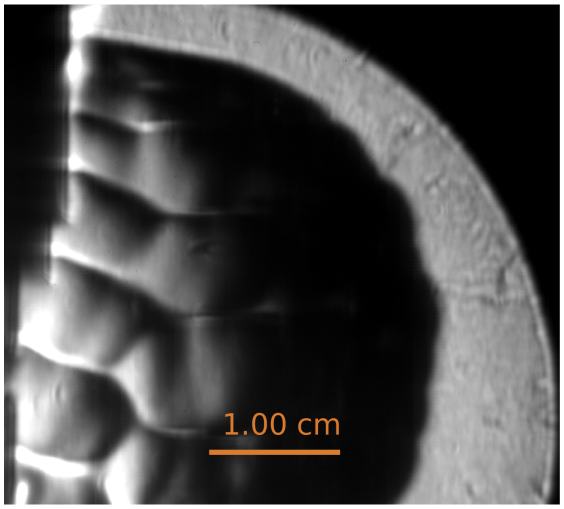


Figure 6.8: Typical Schlieren image of both the location and flame surface structure of the flame propagating from premixed lean mixture ($\phi = 0.63$) through the bubble and into premixed rich mixture ($\phi = 1.4$) after propagating 3.18 cm from the ignition point.

Stratified Flames: Rich into Exceedingly Lean Flame

To investigate the claim that stratified flames can propagate into lean mixtures, a lean mixture of equivalence ratio 0.5 was prepared and rigorously tested to confirm it was unignitable with a homogeneous premixed setup. Subsequently, a stratification layer setup was used to observe a stratified flame propagating from a rich flame ignited in the bubble and into the severely lean mixture.

Figure 6.9 shows a typical flame surface when propagating from a rich mixture into the severely lean premixed mixture following the flame propagating 3.18 cm from the pin ignition center. Similar to Figure 6.5, severe cellular structures were noted in the flame following passage through the stratification layer. As with the rich to lean case, the cellular structure subsided as the flame continued into the severely lean mixture before the flame halted propagation. To highlight and best isolate this effect from the bubble's impact, a thin bubble was made separating the rich and exceedingly lean mixtures. The observed popping of the bubble had significantly limited influence of the flame surface as the bubble popped near the start of ignition and yet cellularities were noted as the flame propagated through the stratification layer.

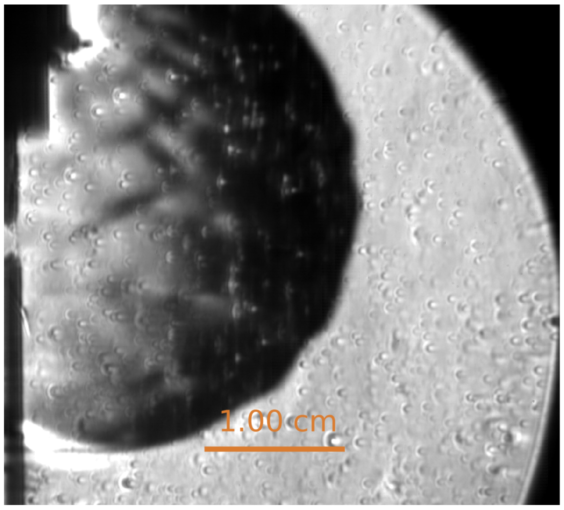


Figure 6.9: Typical Schlieren image of both the location and flame surface structure of the flame propagating from premixed rich mixture through bubble and into premixed severely lean mixture after propagating 2.52 cm from the ignition point.

Further investigation of the cellular structure lead to observing the flame surface of a stratified mixture from a rich mixture into air with cellularities noted in all cases. A typical Schlieren image of a stratified rich to air mixture setup following flame propagation of 2.52 cm from the ignition point can be seen in Figure 6.10.

To evaluate the influence of the mixture composition gradient on the flame front propagation speed, Figure 6.11 shows a typical speed profile of a stratified mixture case where a

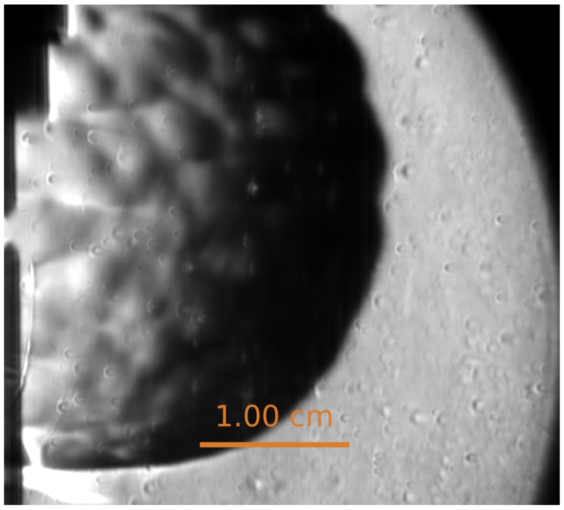


Figure 6.10: Typical Schlieren image of both the location and flame surface structure of the flame propagating from premixed rich mixture through bubble and into air after propagating 2.52 cm from the ignition point.

flame propagates from a rich mixture through a bubble and towards a mixture that is too lean to ignite with the setup. A mean of the rich to air setup is also plotted for comparison.

In both the typical and the mean aggregated flame surface speed, Figure 6.12, the flame appears to burn into the exceedingly lean mixture. It is important to note that the flame surface leading front was tracked and appeared to burn into the lean mixture, however the flame speed dropped as the flame reached the stratification layer and buoyancy effects started to play a role. The buoyant forces distorted the flame shape with minimal change in the overall area observed in the frame.

To further evaluate the location of the flame when the flame stopped burning into the unburned mixture, an evaluation of when the flame halted was taken as a last burning location. The flame location past this point was deemed due to buoyancy effects that artificially tracked the flame forward. When the flame stopped burning downward and the rate of the flame surface movement was no longer nearly uniform in all directions, the frame was marked and the location the flame had progressed was used as the last flame location. However, even accounting for this apparent ending flame location, the flame progressed further than in the rich to air cases.

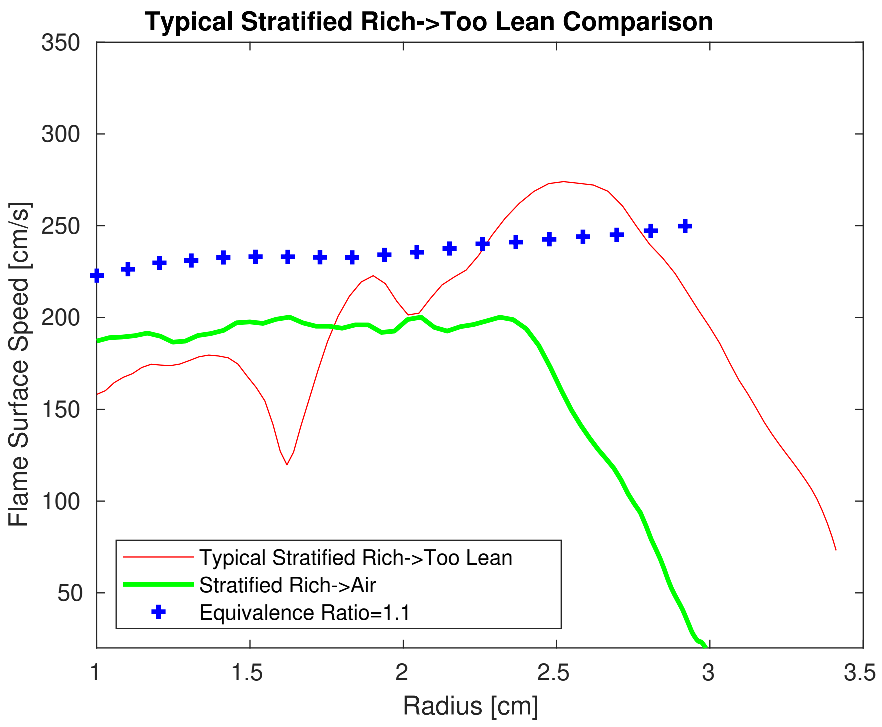


Figure 6.11: Typical flame surface speed of rich to too lean to ignite compared with equivalence ratio 1.1 and a stratified mixture of rich to air. The first two dips and first rise near 2 cm attributed to the difficulty tracking flame as it approached the bubble.

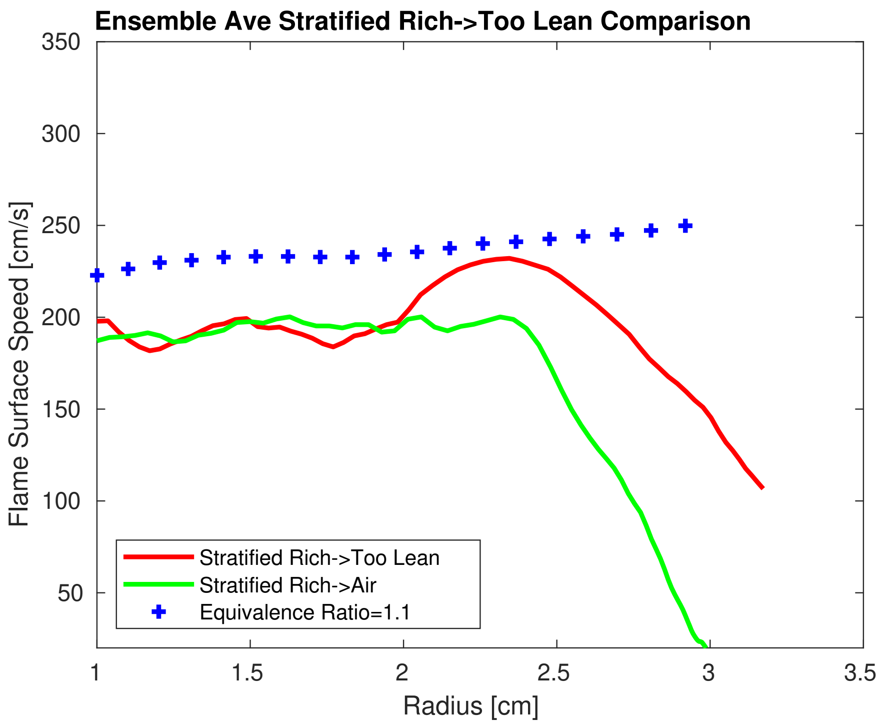


Figure 6.12: Aggregated flame surface speed of rich to too lean to ignite compared with equivalence ratio 1.1 and a stratified mixture of rich to air.

Flame Propagating through an Applied Electric Field

As noted earlier, limited experimental work on the influence of an electric field on flame surface speed was performed and only a proof of concept was developed and tested for later combination. Those preliminary findings are presented below.

A number of applied voltages to the meshes in the electric field setup were tested and it was found that both the positive and negative electric voltages influence the leading flame surface speed. The flame surface was notably distorted and the leading flame front propagated significantly faster than for a similar flame without an applied electric field. Figure 6.13 demonstrates a typical case for an equivalence ratio of 1 for two different applied voltages compared to no applied voltage.

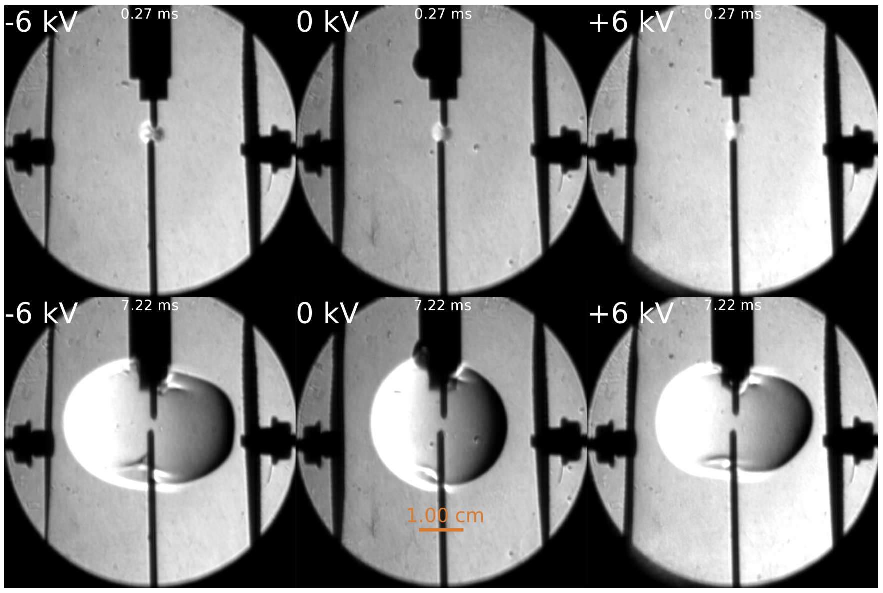


Figure 6.13: Homogenous mixture in applied electric field at two separate times for a negative applied voltage of -6kV (left), no applied voltage (center), and positive applied voltage of 6 kV (right) with the chamber and ignition pins grounded following ignition.

Flame front propagation speeds were noted to increase significantly compared with the no applied voltage case and on the order of 50% in the normal direction of the flame front. However, due to the severe change in shape, the laminar flame speed was difficult to determine and comparisons were performed with propagation speed alone. Additional leading edge flame propagation speeds and further distortion of the flame were noted for higher applied voltage.

Note on Combined effects of Stratified Flames in the Presence of an applied Electric Field

Given the challenges and instabilities found in the stratified field alone in addition to the electric field's propensity to pop the bubble at the applied voltages, extremely limited cases were run. Furthermore, the bubble was popped as the high voltage system was turning on, distorting the stratification layer prematurely before the system had enough time to equilibrate the voltage.

6.2 Numerical Findings

In addition to the experimental findings studying nearly spherical flames, idealized planar and spherical flames were investigated to better understand the experimental factors. Both homogeneous and stratified cases were simulated using a number of models and are compared with the experimental findings below. Since fully modeling the bubble would take enormous computational resources, premixed mixtures were used with stratification modeled as a near step change in equivalence ratio.

Homogenous Flame Propagation in Quazi-one Dimensional Model

To better account for the water that is present in the chamber when a soap bubble is present, PREMIX was used to account for the deviation in the flame speed for comparison with the literature. The adiabatic flame speed for the planar flame under dry and worst case saturated water conditions were determined. Saturated water conditions resulted in flame speed reductions of 5-10% that varied based on equivalence ratio, as seen in Figure 6.14.

Homogenous Flame Propagation in One Dimensional Spherical Homogenous Flame Propagation Model

In addition to using a quasi one dimensional model to validate the homogeneous mixtures and compare with laminar flame speed, an axisymmetric flame code (ASURF) with the GRI 3.0 detailed mechanism was used to compare with the experimental values. Laminar flame speeds of the experimental homogeneous mixtures are compared with the one dimensional code and agree well, seen in Figure 6.15. The flame surface speeds and therefore the laminar flame speeds predicted in the simulations were noted to be somewhat faster than observed in the experimental conditions of this work. However, this was also observed when comparing the simulation with many experimental values in the literature with reasonably good agreement achieved.

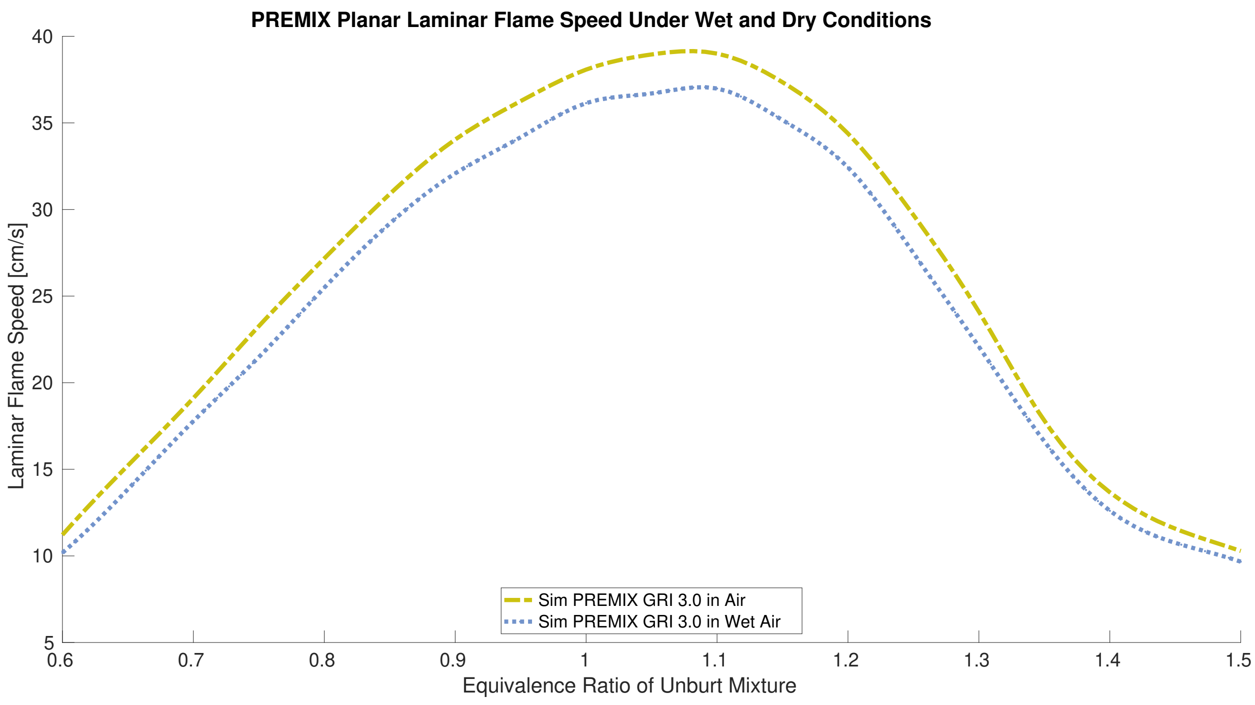


Figure 6.14: PREMIX planar laminar flame speed of methane-air at various equivalence ratios under wet vs. dry ambient conditions.

Homogenous Flame Propagation with Three Dimensional Code

Simulations of homogeneous flames were preformed using a three dimensional CFD code (CONVERGE) with coupled chemistry to undergo an initial investigation of a possible cause to the instabilities noted in the experiment. The boundary and initial conditions are laid out in Section 5.3, with a quartered constant volume chamber divided on the lines of symmetry. An example of the mesh and grid is referenced in Figure 5.1 using AMR to ensure enough grid points near the flame. The rich ($\phi = 1.4$) mixture throughout the chamber was initially at 300 K and 1 atm and ignited by depositing 300 mJ over a 3 mm radius sphere over 2 ms and the resulting flame allowed to propagate.

In particular, the flame structure was of particular interest in the stratified cases as the behavior of the stratified experiments deviated with the expected results following recent literature. While the flame could not be resolved to the same level of refinement as the one dimensional flame code due to memory and computational effort scaling, homogeneous results provided a way to observe the flame surface structure under ideal conditions and to see if it remained smooth and to compare this to a stratified case. The flame surface front can be nearly located by H species as these radicals are short lived and only exist near the reacting gas, however the flame surface temperature represents an integrated effect of the varying chemical reactions. The limited distortions in the flame surface front were observed in the iso-temperature surface near the equilibrium temperature of the gas mixture for the

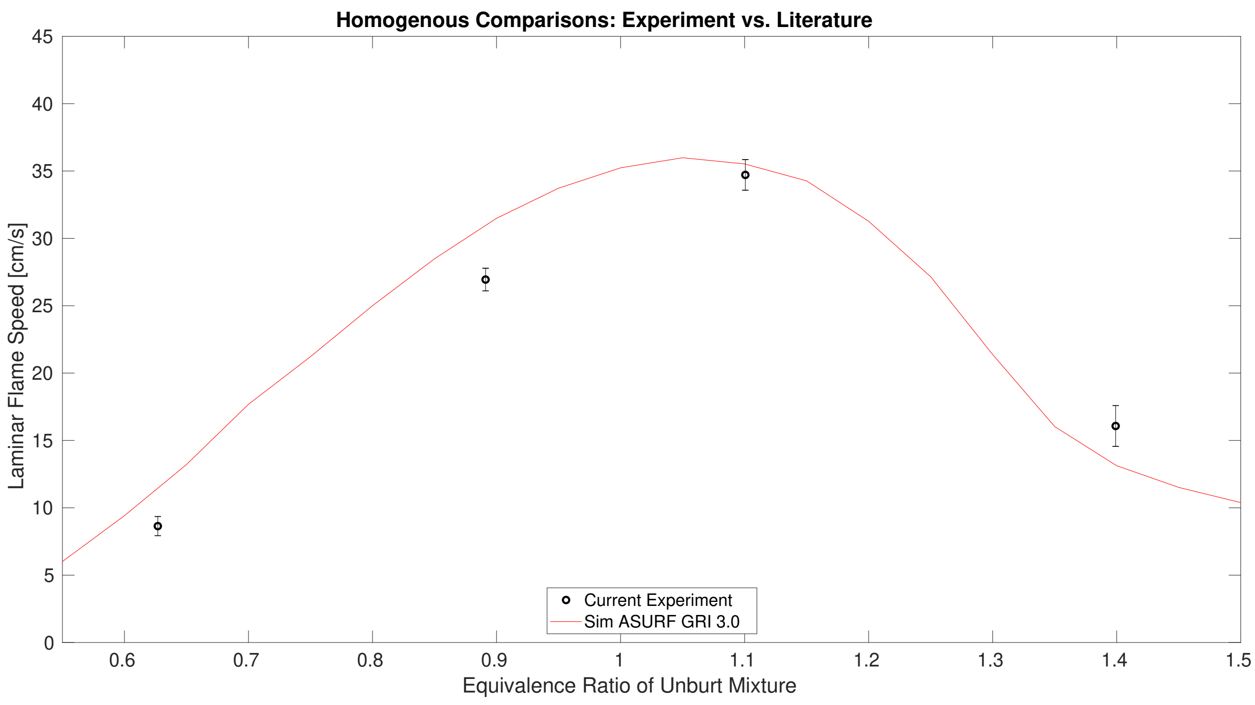


Figure 6.15: Typical homogeneous experimental laminar flame surface speed vs. one dimensional spherical flame propagation in good agreement.

homogeneous case. The flame surface was relatively smooth for the homogeneous flame as can be seen for a typical rich flame in the iso-temperature surface plot in Figure 6.16.

Note on Three-Dimensional Confinement and Heat Transfer Effects

In reviewing long times of the three dimensional results, confinement effects started to play a role later in the flame development as the flame began to interact with the walls of the constant volume chamber. However, these effects were noted to take place beyond the optical space and after the flame had already passed beyond the view of the window and therefore confinement effects were not considered a significant factor in the flame speed calculations from the optical setup.

Stratified Flame Propagation in One Dimensional Spherical Stratified Flame Propagation Model

Various numerical tools were used to evaluate and compare to the experimental findings. In particular one and three dimensional tools were used to evaluate the flame dynamics and structure.

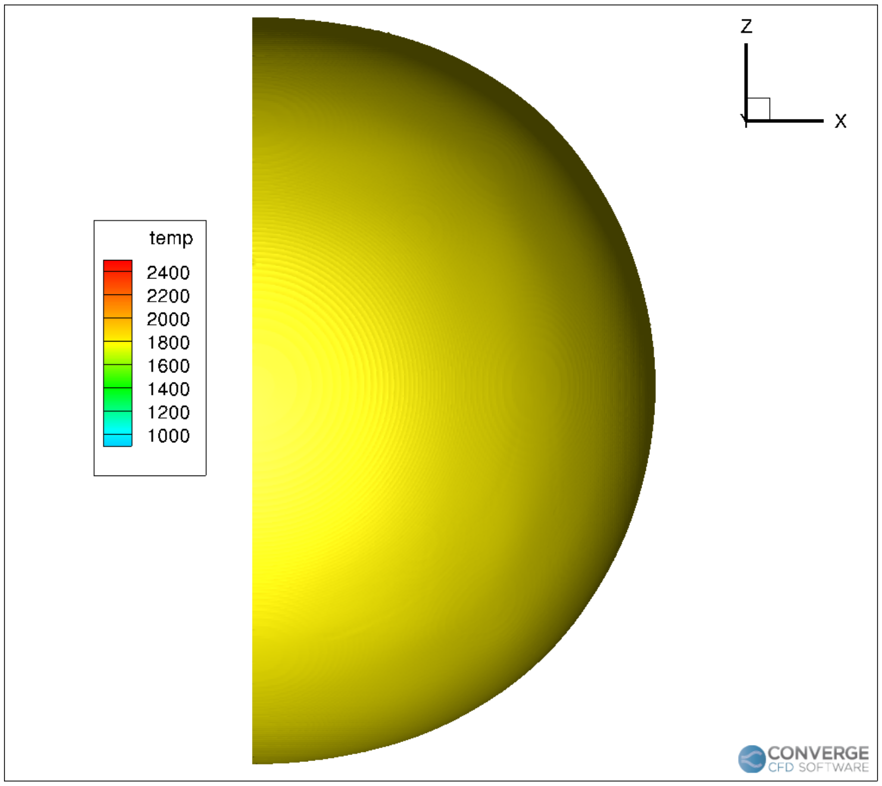


Figure 6.16: Three dimensional flame iso-temperature surface profile near the equilibrium temperature of homogeneous rich flame within the ARM refined region after 25.2 ms.

Expanding on work done by Shi, one dimensional stratification results were compared to the experimental work. As with the homogenous cases, the numerical results predicted higher flame speeds than were observed in the experimental work. A comparative plot of the experimental stratified rich to lean case with a series of one dimensional spherical stratified flame speeds can be seen in Figure 6.17 and further discussions are provided in the following chapter, Chapter 7.

Stratified Flame Propagation in Three Dimensional Stratified Flame Propagation Model

Three dimensional modeling of the constant volume chamber and subsequent flame dynamics were investigated. Using the same geometry of the homogeneous cases, the ideal stratified case of a flame propagating from a rich ($\phi = 1.4$) to a lean ($\phi = 0.65$) mixture was investigated due to the cellular structures noted in the experiment, following the otherwise identical initial and boundary conditions as referenced in Section 5.3. If the non smooth flame surface structure were due to the stratification, it would not be inherently obvious with the lower dimensional codes performed by others in the literature.

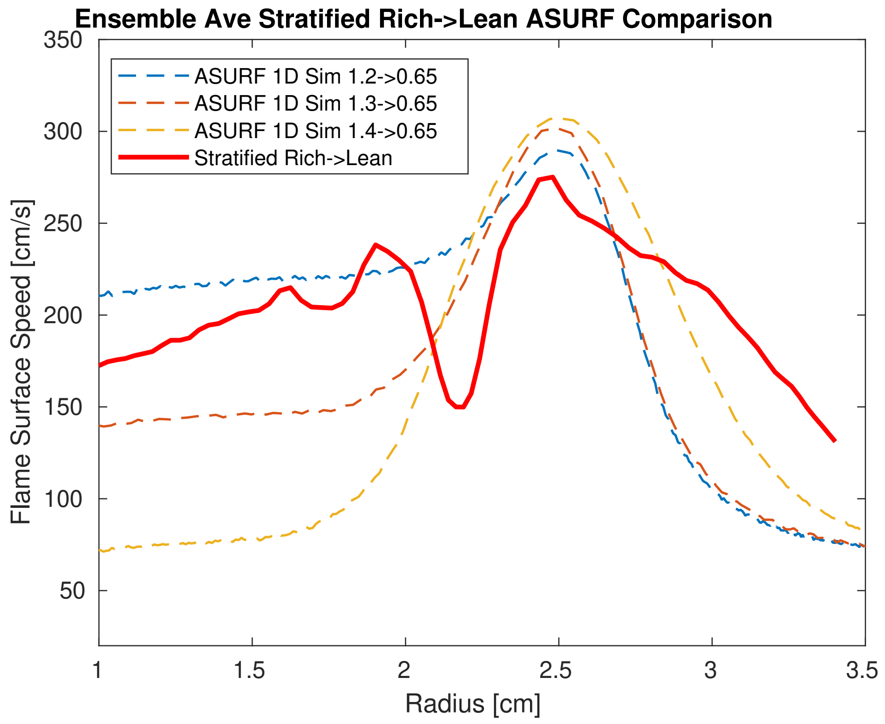


Figure 6.17: Aggregated experiment stratified flame burning from rich into lean mixture compared with one dimensional spherical simulation.

Care was taken to ensure the mesh was as fine as possible, and an adaptive mesh refinement scheme was used to ensure the grid was focused near the relative combustion driving factors of temperature and radical concentration. The mesh was modified to ensure a reasonable refinement of 1.25E-1 mm through the stratification layer and equivalent parameters set as the homogeneous case as the flame progressed through the stratification layer. The equations, boundary conditions, and initial conditions are laid out in Section 5.3, with the same dimensions as the experimental work but with the constant volume chamber quartered and divided along the lines of symmetry. An example of the mesh and grid is referenced in Figure 5.1 using AMR to ensure enough grid points near the flame. The rich ($\phi = 1.4$) mixture within the bubble zone as well as the lean ($\phi = 0.65$) mixture throughout the rest of the chamber was initially at 300 K and 1 atm and ignited by depositing 300 mJ over a 3 mm radius sphere over 2 ms and the resulting flame allowed to propagate. A few particular features were noted different than the homogenous cases, besides higher temperatures at the flame surface during the transition into the lean mixture.

As with the homogeneous flame, the flame surface front can be nearly located by the H species and temperature near the equilibrium temperature. However, the temperature of the flame in the stratified case increased as it passed through the stratification layer. Interestingly as the flame transitioned through the stratification layer, a set of irregular structures formed on the surface of the flame front that were not present before when in a purely rich mixture. With the first pass of the simulation with less refinement, the flame front had irregular structures that was initially attributed to the limitations of resolving the computational grid of the flame. However, both with the more refined mesh and when compared with the experimental results, the findings appeared to agree well.

The refined version of the three dimensional flame front of the iso-temperature surface plot as the flame transitioned into the stratification layer can be seen in Figure 6.18. While the adaptive mesh refinement grid allocation was not focused on resolving the inner region of the burned gas, the outer layers at the reacting region are clearly irregular and appears as cells and surface disturbances similar to the onset of the cellular structures noted in the experiment.

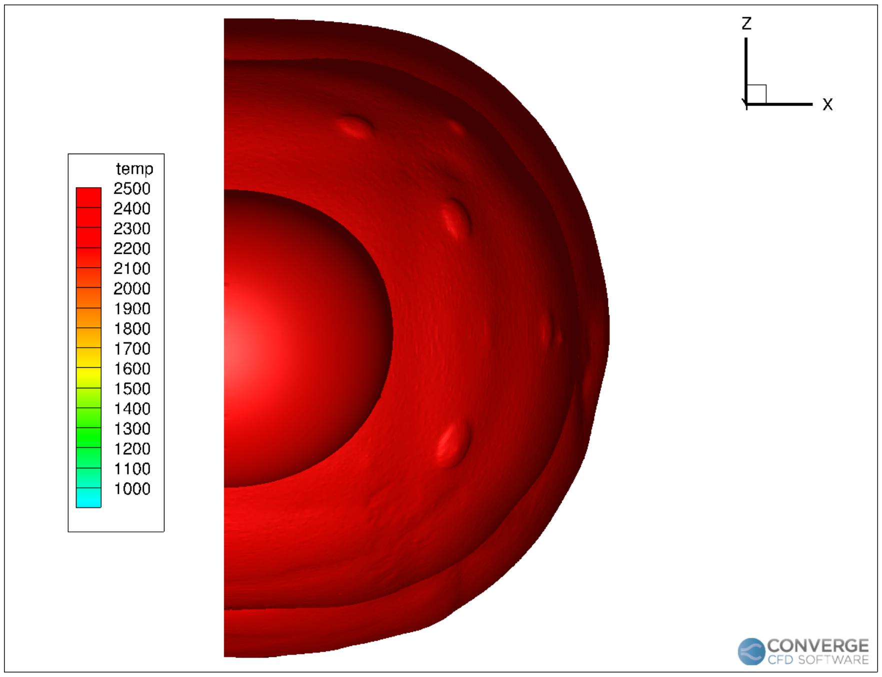


Figure 6.18: Three dimensional flame iso-temperature surface profile near the equilibrium temperature of stratified rich to lean flame within the ARM refined region after 25.2 ms, with middle surface indicative of the non-smooth flame surface front.

Chapter 7

Discussion of Experimental and Numerical Findings

A variety of homogenous premixed mixtures were made and used in either homogeneous or stratified setups. The homogenous premixed laminar flame speeds determined from the experiment agreed well with past work in the literature. In addition, the flame surface speed of a homogeneous premixed methane-air equivalence ratio corresponding to the fastest deflagration speed was experimentally measured and compared with the stratified cases. A variety of noticeable differences arose between the stratified mixtures and the homogeneous mixtures tested. Overall in the experiments three key differences arose.

The stratified flames exhibited a faster flame surface speed within the transition region than capable with a homogeneous premixed mixture alone. Emissions differed with the stratification as compared with an equivalent homogeneous mixture. The flame surface structure was noted to be distorted in the transition region. Lastly, the flammability limit appeared to be extended and is further discussed in this chapter.

7.1 Stratification Layer Flame Speed Enhancement

Simplifying the flame structure as done in the past by others, and assuming that the flame propagation occurs nearly smoothly and that the flame surface speed enhancements are only due to the stratification layer, an overall speed enhancement was noted. Experimentally, the flame surface speed of the stratified cases burned at a significantly faster rate through the transition region than was possible with a homogeneous mixture alone by a peak increase of 17 % over a maximum for a corresponding radius of the max homogeneous flame case ($\phi = 1.1$), as seen in Figures 6.6 and 6.7. Accounting for the water content, the stratified flame speed enhancement is nearly 29% increase.

Experiment Compared with one dimensional spherical stratification

As can be seen in Figure 6.17, the numerical one dimensional model of a stratified case of similar geometry compared with the experiments show reasonable agreement. However, there were three main regions of interest that indicated a variety of differences in the experimental setup than the idealized one dimensional simulation results.

The first region of interest is in the initial rising slope of the experiment from 1 cm to the interface of the expanded bubble near 2 cm. Interestingly, the trend of the flame speed rises at a similar rate as that of a stratified transition from equivalence ratio of 1.3 to an equivalence ratio of 0.65. The initial rise in speed occurs before the flame front reaches the bubble and at a rate much greater than seen in the simulation. This is attributed to a finite amount of permeability and a diffusion layer through the bubble. This diffusion occurs from both the finite time from blowing the bubble and ignition as well as from the enlarged thinning bubble from the sudden expansion of the burned gases. The two dips in flame surface speed along with an intermediate rise are attributed to the difficulty in tracking the flame surface when the flame neared the bubble, and are therefore not physically meaningful.

The second region of interest is in the transition region as the flame propagated through the stratification layer. The experimental stratified flames exhibited a noticeable flame speed increase above that reached with a flame propagating through a homogeneous premixed mixture of equivalence ratio of 1.1, the highest deflagration speed for a methane-air premixed mixture, achieved in the setup. This also agrees well with past literature and is due to the back support of both the thermal diffusion and preferential hydrogen ahead of the flame. While there are differences in the extent of the flame surface speed enhancement, the discrepancy in the stratification transition region from the one dimensional simulation can be attributed by two factors. For one, the simulation did not model the bubble and the resulting stratification layer in the same way as was present in the experiment. The experiment had a finite amount of water vapor in the transition layer that was not modeled in the simulation which accounts for a portion of the drop in flame speed, as can be seen in laminar flame speed differences in wet and dry mixtures shown in Figure 6.14. In addition, the stratification layer modeled in the simulation may have been sharper than found in the experiment, subduing the stratification speed enhancement. Simulations have assumed a very sharp gradient around 1 mm, while the exact thickness is unknown in the experiment and likely much larger. Larger stratification thicknesses will reduce the effects of the stratification. Lastly, as the simulated flame speeds were higher in the simulation for all homogeneous cases as seen in Figure 6.15, there is likely an over prediction of the flame surface speed compared to the experimental values for the stratification transition region as well.

The last region of interest is the flame surface speed as the flame transitions further into the lean mixture, and in particular the slope of the flame surface speed. The flame speed drop off has a similar rate as that seen in the simulated stratified transition from equivalence ratio of 1.3 to an equivalence ratio of 0.65.

While the maximum flame speed enhancement is difficult to quantify as the underlying local equivalence ratio in the experiment is difficult to ascertain, excellent qualitative agreement is achieved between the experimental and numerical values after taking into account the differences.

7.2 Emissions

Both unburned premixed tank mixtures as well as exhaust gas of the experiments were collected, diluted with argon, and tested with a gas analyzer. As mentioned previously, due to calibration limitations in the CO analyzer,the exact quantitative value was not available but a quantitative reference was trusted. As can be seen in Tables 6.1 and 6.2, overall the rich to lean stratified cases underwent partial oxidation of the fuel more efficiently than the homogeneous equivalent, however at the expense of additional CO.

Total Unburnt Hydrocarbons

Decreased hydrocarbon emissions were noted in the experiments with stratification layers from rich to lean. However, an increase was noted for the rich to severely lean stratified cases.

For the rich to lean case, the decrease in unburned hydrocarbons indicates a better breakdown of the fuel and subsequent oxidation. This may be due to the back support as noted in the literature as the flame burns. Additionally, as most of the unburnt mixture comes from the rich side of the stratification layer, local mixing is an important factor following flame propagation. After the flame propagates through the mixtures, additional reactions near where the burned rich mixture is located take place as it mixes with the lean mixture that contains unreacted oxygen, thereby it is locally more reactive than in the homogeneous equivalent mixture. In addition, the temperature of this rich region mixing with the leaner burned gas is elevated, resulting in faster mixing and breakdown of the fuel before the reaction is quenched.

For the rich to severely lean cases, a different effect occurred. Most of the unburnt hydrocarbons can be attributed to the severely lean mixture presence as the flame did not fully propagate into this region. Additionally, an increase in the unburned hydrocarbons is due to the reduction in overall chamber temperature from the reduced volume burned by the flame. With this reduced temperature, reactions stop quickly and mixing of fuel and oxidizer occurs slower. Both of these effects support quenching of the reaction zone and lead to increased unburnt hydrocarbons in the exhaust gas. This result was in agreement with work by Galizzi and Escudié[20], where additional CH^* was noted past the oblique flame front.

Molecular Oxygen

In both cases, a significant portion of oxygen remained as expected given that a large portion of fuel was left unburned.

For the stratified rich to lean case as well as the homogeneous equivalent, the overall equivalence ratio was lean. As lean homogeneous premixed flames typically exhibit unconsumed oxygen, this was expected.

Similarly for the rich to severely lean case, the oxygen remained high from incomplete combustion.

Carbon Dioxide

For the stratified rich to lean case, the concentration of carbon dioxide in the exhaust was lower than the homogeneous equivalent. This would indicate the reduction in the oxidation reaction of CO. As the breakdown pathways of fuel first oxidize the carbon atom into CO, it is clear the temperature was not sustained to oxidize the CO into CO_2 through the dry routes given in Equations 2.12 and 2.13. Additionally, with the quenching of the flame, less OH radicals are likely to be present which limits the oxidation step given in Equation 2.11. These incomplete reactions occur more frequently in this stratified cases as the reactions following flame propagation are at a higher temperature but below the minimum required to convert CO into $CO2$ in the presence of molecular oxygen or oxygen radicals. Similarly for the rich to too lean case, the incomplete combustion was noted in the lack of conversion to CO_2. However, with a cooler temperature, less unburnt hydrocarbons were converted to CO.

Nitrous Oxides

For all cases considered, the exhaust gas concentration of nitrous oxides was low, with the lowest value from the rich to severely lean. This result from the rich to severely lean experiments is unsurprising as previously mentioned this mixture prematurely ended combustion, limiting the thermal pathways given in Equations 2.14, 2.15, and 2.16. As much of the NO_x production comes from high temperature breakdown of the nitrogen molecules, without the sustained high temperature of the flame to the gas, extremely low values were expected. Even though the mixture continues to react at an elevated temperature, the thermal NO_x production is not sustained due to the lower temperatures and NO_x is mainly produced only through less efficient pathways such as those referenced in Equation 2.17.

For the rich to lean stratified cases, the decrease in NO_x production is also due to the reduced thermal pathways from the reduced temperature while some reactions continued following flame propagation. Limited NO_x is produced in the rich region as the flame has limited oxygen, limiting the oxygen dependent pathways referenced in Equations 2.14, 2.15, and 2.16, In addition, the correspondingly reduced flame temperature limits the prompt formation pathways referenced in Equation 2.17. As the flame propagates through the lean mixture, the temperature of the flame is lower than that of an equivalent overall homogeneous mixture case. Subsequently, the NO_x formed from the homogeneous equivalent to the stratified case produced more NO_x as the temperature was higher during this flame propagation. This is in partial agreement with past work on NO_x emission by Furuno et. al[19] who noted that the NO_x production was not severely impacted by the stratification.

7.3 Stratified Flame Surface

As previously noted when the flame propagated through the transition layer and soon after, the stratified flame experiments exhibited a flame surface structure that deviated from the expected smooth surface. Although there were instabilities present in the flame due to a variety of fluid dynamic effects from a popping bubble, there were pronounced differences in the flame surface that the bubble popping alone could not account for. The initial flame development in the stratified rich to lean cases approached the bubble slowly with a proceeding flame-bubble interaction similar to that seen in the flame surface structure of the homogeneous premixed rich and homogeneous premixed lean cases. It was expected that the flame surface would similarly be smooth as seen following propagation through the transition layer, however this was not the case. In particular, cellular structures were noted to form in all of the rich to lean mixtures, rich to air mixtures, as well as the lean to rich mixtures, as noted in Figures 6.5, 6.9, 6.10, and 6.8.

As the cell structures in the flame were observed soon after the flame passed through the initial transition region, special attention was paid to this transition layer. During severe bubble popping as in the case of thick bubbles or with very fast propagating flames, ripples and in some cases cell structures were noted to briefly appear in the flame structure during the transition between mixtures before self-healing and returning to a near-spherical shape. While, the transient instabilities seen in the stratified cases can be partially attributed to a variety of fluid dynamic instabilities, these cases appeared to exhibit additional inherent instabilities that fluid dynamics alone could not account for. Unlike the homogeneous cases with a bubble, the cell structures in the stratified cases appeared regardless of how vigorous the bubble was popped and regardless of when the bubble popped. Interestingly, the cellular structures appeared for all stratified cases including thin bubbles, which had minimal impact on the homogeneous premixed cases.

Idealized Flame Structure Assumption

Stratified flame propagation in past work has mostly focused on the overall burning rate of the flame using either pressure rise rate or light detection networks to deduce the propagation speed of the flame. Such methods employed rely on several key assumptions regarding the surface of the flame, namely that the flame exhibits a smooth and regular surface and that idealized models such as perfectly spherical flame or planar flame accurately models the flame dynamics. However, such methods ignore the possible relevant flame structure which may play a significant role in the determination of the flame speed.

Analysis in this work was first performed assuming a perfectly spherical flame, as has been done with similar experiments in the literature to quantify the spherical flame propagation. As, this work has mostly relied on using visual methods to track the flame surface, significant differences were observed than expected. While a smooth flame transition between the decreasing equivalence ratio gradient was expected, various unexpected structures were observed for all stratified cases explored. The differences may be accounted for by not only hydrodynamic instabilities induced by a moving surface sheet but also thermo-diffusive instabilities inherent in a stratified flame.

Following review of the flame surface structure and relevant possible instabilities in these cases, several mechanisms stuck out as likely culprits. Those mechanisms inherent in the bubble's presence and those inherent in the stratification itself. In particular, the velocity shear layer induced by a popping bubble, the added compositions from the bubble, in addition to thermo-diffusive instabilities.

Influence of the Soap Bubble

Velocity Induced Instabilities

To mitigate this inherent disturbance caused by a popping bubble that could not be appropriately modeled numerically, varying soap mixtures were made with a variety of different bases and their effectiveness was partially evaluated based on optimizing the mixture to have a limited impact on the flame and stratification layer when the bubble popped. At first, a very thin bubble was desired as minimal impact on the flame propagation behavior was observed. However, it became apparent in subsequent experiments that this sacrificed the impermeability of the soap mixture, increasing the permeability to an undesirable level. This permeability reduced the equivalence ratio within the bubble in the initial rich to lean cases as well as the stratification layer gradient by significantly increasing the stratification layer thickness.

To confirm this, ignition tests with a non ignitable severely rich to air stratified setups with this bubble soap mixture evaluated the extent of the combined permeability and diffusion of fuel through the bubble and the air into the bubble. Results of these tests indicated a quick reduction of equivalence ratio within the bubble to the greater chamber on the order of seconds. Therefore, a trade-off was necessary between the soap mixture permeability and velocity induced effects on the flame.

Increasing the water content had the desired effect of decreasing the permeability with the side effect of increased velocity shear layer disturbances. The soap solutions with the most water made the thickest bubbles as noted in the Schlieren images, which were noted to have the largest impact on the flame. This influence of the bubble was also observed depending on the amount of fluid and the soap solution used while exploring different soap solutions. In particular, depending on the surface tension when the bubble popped, the bubble pushed the fluid of mixture to varying degrees. This effect was limited to some degree by the selection of soap solutions used. Several methods of bubble popping were attempted, but were not successful for the final soap solution attempted. In addition, the dynamics of the bubble popping and subsequent influence on the flow and mixture stratification field occurred quickly, obstructing the known mixture stratification. Image processing was undertaken to account for the slight bulk fluid flow induced by the bubble popping dynamics that was allowed for in the trade offs of the experiment. Despite the fluid dynamic movement of the unburned gas and flame, this was determined to have limited impact on the flame surface structure and flame surface speed noted in this work.

It is important to also note that the speed at which the flame surface speed progressed within the bubble directly impacted the popping dynamics of the bubble. As most of the cases considered either had a rich or lean flame within the bubble, the approach of the flame had minimal impact to the popping dynamics of the bubble. However, for mixtures that propagated faster, as in the case of a near stoichiometric mixture, the flame appeared to have significantly more influence on the bubble popping as these flames are characterized by a quick flame propagation, a resulting faster pressure rise, and a higher and faster expansion of the gases. Due to less time for evaporation and the fluid stretch of the expanding bubble, the thickness of the bubble when it popped was thicker. With a thicker bubble present at bubble breakdown, the extent of velocity shearing was greater. Therefore the extent at which flame passing through this region had an increased impacted by this fluid instability. Therefore, it was reasoned that the cellular structure of the stoichiometric case with a bubble was due to the coupled effect of this instability and the fast propagating leading up to the bubble.

However, for the stratified cases, the flame propagated slowly as it approached the bubble with significantly less overall volumetric expansion and pressure rise rate. Therefore, the coupled effect of the flame and bubble was limited, and provided a limitation on the extent of the velocity shear layer instability.

Undoubtedly the bubble played some role in the instabilities present as the flame propagated, however the effects of the bubble was severely limited in a set of experiments to tease out its effect. By selecting the thinnest bubble possible and sacrificing permeability and diffusion temporarily, it was seen to confirm the cellular structure appeared. Therefore, while the exact extent of the velocity shear layer's effect on the flame surface structure was not directly modeled, it was experimentally explored and indicated a decoupled effect from the fluid dynamic instabilities.

Soap Bubble Mixture Contributions

An alternative possible explanation to the cellular structure may be partially attributed to either the water or the hydrocarbons embedded within the aqueous solution, the saponified Olea oil and glycerine. For reference, the contents of the soap solution is referenced in Table 6.3.

In past work with heavier hydrocarbons, instabilities were noted to form in the flame surface. The hydrocarbon gases that have been tested prior in the literature may appear in the pyrolysis reactions of the heavier hydrocarbons within the soap solution.

For the hydrocarbon contribution, the cellular structure was not observed for any of the wet homogeneous cases run. If the cellular structures were from the soap solution, similar strong cellular structures would be in all of the homogeneous cases including those ran homogeneously with the bubble. This was not the case as seen in Figures 6.2 and 6.3.

Furthermore, if the addition of hydrocarbons from the soap bubble played a key role beyond its vapor saturation in the cellular structure formation, this would have meant that the cellular structure would scale with the concentration of hydrocarbons within the soap mixture. This type of cellular scaling was not observed with the concentrations or soap base solutions tested. Additionally, the concentration of hydrocarbons with the soap solution selected was quite minimal with a small fraction of both the mass and volume coming from the soap solution.

For all of these reasons, it was determined that the hydrocarbons within the soap were unlikely to significantly contribute to the cellular structures observed in all the stratified cases.

While the hydrocarbons within the soap may have extremely limited impact on the flame, the water content is another possibility. This would indicate that the cellular structures are due to either water vapor or water droplets. Water vapor has not been reported to cause cellular structures, nor was it seen when homogeneous mixtures were burned in wet environments or with a bubble in this work. In addition, in this work the flame exhibited the cellular structures long after the bubble popped in many cases for all of the stratified cases with enough time for large water droplets to move away from the transition region. Therefore, the influence of the water was ruled out as a possible main contributing cause.

Therefore, the influence on the flame structure from the mixture components of both water and additional hydrocarbons was determined to have limited impact on the structure of the flame.

Thermo-diffusive Instabilities

While the velocity shear layer may play a large role for severely popping bubbles, as in the case of a stoichiometric mixture quickly expanding and exploding the bubble, additional factors were explored to account for the cellular structure of slower flames that had less vigorous or nearly non-existent impacts from the popping bubble within the flame surface path.

As thermo-diffusive instabilities can arise when there is a miss match of thermal and mass diffusion as well as under flame stretch. Either the unequal heat transfer from the exhaust gas to the unburnt mixture or the preferential diffusion of hydrogen species would need to play a role. Given the work from Shi et. al[54], one dimensional simulations of spherical expanding stratified flames have been shown in simulation to exhibit a strong preferential diffusion of hydrogen species ahead of the flame front as a stratified flame is propagating from a rich to lean region.

Cell structures were observed as the flame propagated through all of the stratified cases explored, in addition to the speed enhancements. These structures were mostly attributed to the back support of hydrogen species from the rich flames that temporarily induce thermo-diffusive instabilities. As presented in past work, the hydrogen species diffused ahead of the thermal diffusion front, indicating a non-unity Lewis number.

In stratified rich to lean flames, the instabilities is accounted for by the preferential diffusion of hydrogen noted by Shi in one-dimensional studies but the stability of the flame could not be uncovered in these one-dimensional flames. This work has shown both in the extensive experiments and the limited exploration of numerical three dimensional modeling that flame instabilities may arise at the interface of the stratification layer of a rich to lean flame, and in particular cellular structures as can be seen in Figure 6.18 but not in in the rich case as in Figure 6.16. The three dimensional modeling of spherical flames in the constant volume chamber have indicated a flame structure similar to that of a grouping of cells, first attributed to numerical meshing effects but later shown to appear even with a refined mesh.

Experimentally, all of the rich to severely lean stratified flames and rich to air stratified cases exhibited the cellular instabilities in a similar way as the rich to lean cases. Interestingly, stratified lean to rich flames in this work were also noted to exhibit cellular flame surface as seen in Figure 6.8. This lean to rich case was attributed less to hydrogen preferentially diffusing ahead of the flame and more to the thermal conductivity differences of the burned gas and the unburned gas as has been noted in planar flames in past work by Einbinder, et. al[17].

Comparison with Past Work

It is also important to note prior work on methane-air flames with the addition of hydrogen disrupts the stability of an otherwise spherically smooth outwardly propagating flame.

Initial flame instability work by Markstein[37] and Behrens[4] found that addition of hydrogen to methane flames lead to cellular structures. Work of Einbinder[17] saw that the lean to rich planar flames exhibit instabilities due to the non unity lewis number as the thermal diffusion is limited in transferring the thermal energy into the flame front.

More recently, Hu et. al[26] noted in their work that the addition of hydrogen increased the propensity of a methane-hydrogen-air flame to exhibit thermo-diffusive instabilities and the flame surface appeared similar to that of the well studied lean hydrogen flame. Lean hydrogen flames exhibit cellular structures similar to that seen in this work.

As cellular structures were noted in the experiment during the transition region, this work partially focused on comparisons with past numerical work and the notation from past experimentation on stratified flames.

Past numerical work has not indicated such instabilities in stratified flames, although this may be attributed to the fact that the numerical simulation and model work on stratified flames has mostly used either theoretical models or idealized one-dimensional models. The one-dimensional simulation was unable to confirm the existence of additional instabilities as the cellular structure is a three dimensional effect that would not manifest itself in an idealized one dimensional code. As such, these models are unable to fully account for various multi-dimensional effects including the onset of a variety of instabilities such as the effect of thermo-diffusive inequalities on the flame structure.

Past work on identifying the flame structure disturbance has been impeded by various experimental setups. In reviewing the experimental literature, this cellular structure phenomenon is either supported or not excluded by past experimental work on stratified flames. Other several past experimental works noted or showed images of wrinkles in the flames, but was only attributed to the limitations of the experimental setup and no further investigation was sought.

Past spherical flame experiments similar to this work by Ra[44] briefly mentioned the existence of a disturbance from the Schlieren images but did not present an investigation or further discussion. Additionally, even with highly advanced species and flow measurements in the work by Galizzi and Escudié[20], instabilities developed but were attributed to the flow conditions yet were noted to initialize in less than 0.5 ms. The time frame is indicative of either mass or thermal transport related instabilities as the flow was laminar and the fluid dynamic effects will likely take longer to develop. Later imaging of the flame structure by Schmidt [51] noticed wrinkling in their experimental work on stratified planar flames observed on a flat plate burner with chemiluminescence but all of the wrinkling were attributed to the experimental setup.

A large portion of the past work did not provide a description of the flame structure. For instance, early work by Karim et. al[31] visualizing the flame using Schlieren made no mention of the flame structure and given the choice of vertical tube, the experiment likely suffered from Rayleigh-Taylor instabilities that would have prevented the accurate accounting of the flame structure in the Schlieren images.

Contrary to prior work, additional focus has been paid to the structure of the flame in this work. Additionally, this work does not have the same limitations in the previous experimental setups, and the spherical flame is free to propagate through a stratified layer mostly undisturbed.

It is possible however, that the size of the flame at the transition was such that the instabilities took a much greater impact than past work. As stretch has been known to provide a support to an otherwise unstable flame, the stretch rate when the flame in this work reached the transition may not have been enough to suppress the instabilities. Full exploration of different size bubbles was not possible in this setup to confirm the unanimous nature of the cellular structures, but it is believed that it is a likely tripping mechanism in all stratified flames.

7.4 Evaluation of Flammability Limit

There is no consensus of whether a stratified flame will assist in propagating through a lean flammability limit if it is back supported by a rich flame. Some previous experiments have indicated this to be the case, however in this work the extent of this was not definitive in post-processing or subsequent modeling. Numerical work has also shown this to not to be significant. Simulated stratified spherical flames propagating to the lean flammability limit do not continue to propagate through the mixture significantly as noted by Shi[54].

In both the typical and the mean aggregated flame surface speed, seen in Figures 6.11 and 6.12, the flame appears to burn into the exceedingly lean mixture, appearing to be confirmed by the additional manual exploration of the flame halt location. However, there are several factors that make it difficult to ascertain the exact extent of the lean burning, even while the flame progressed further into the mixture than ideally allowed with a step change in mixture concentration. Three areas of likely contribution include the original mixtures used, the finite diffusion time, and the difficulty in tracking the local equivalence ratio as the flame approaches the mixture.

Discrepancies in Mixture

Possible discrepancies may be due to mixture preparation and inadequate determination of the equivalence ratio near the lean limit. Past work has relied on extremely limited mixing times for premixed mixtures. However, the burning rate of flames far from stoichiometric becomes increasingly sensitive to equivalence ratio and the short times used in past work may be inadequate to ensure no local zones of more highly rich zones. As highlighted by Chen[8], methods in the literature may be highly sensitive to the method of mixing. Rigorous procedural methods were undertaken in this work to target mixtures and ensure proper tank mixing.

Tracking of Burned Zone

The flammability limit appeared to extend into the exceedingly lean mixture, however it is important to note that tracking of the flame became questionable as the flame became exceedingly non spherical both in surface structure as well as in overall shape. The side projection as seen in the Schlieren became non circular as the flame slowed as it reached the interface with the severely lean mixture and buoyant effects distorted the progress of the flame. Additionally, the cellular structures made it difficult to evaluate expansion effects. Without an appropriate method to decouple the dependence of the flame surface speed on this fluid dynamic effect, evaluation of the burned zone is tenuous at best. Unfortunately, the resulting pressure rise during the flame propagation within the optical window was near the pressure equipment minimum threshold, so it couldn't be used to discern the exact amount of burning with a high degree of confidence.

Finite Diffusive Time

It is important to note that given the limited diagnostic tools available in this work, the mixture diffusion layer was difficult to measure in this setup and may be a contributing factor. Furthermore, since the existence of diffusion and permeability influences the mixture concentration, the exact equivalence ratio may not be experimentally determined with the methods used and available in this and past work on stratified flames. The finite amount of diffusion that occurs either from the permeability of the bubble once blown and as it expands and thins, in addition to the convective and molecular diffusion that occurs as the flame approaches the stratification layer distorts the true local equivalence ratio before the flame reaches the lean severely lean equivalence ratio interface. Therefore, while the second mixture within the chamber may have initially been too lean to propagate through, the flame deflagration occurs over a finite time while the stratification layer mixes. By the time the flame reaches the end of the stratification layer, the end gas mixture has already somewhat mixed and it is possible to underestimate the equivalence ratio that the flame sees locally.

Chapter 8

Concluding Remarks

8.1 Summary of Findings

An experimental setup was constructed to test and explore the effect of both stratification and electric fields on spherically expanding flames of various methane-air concentrations. A majority of the focus of this was on the stratification exploration. A number of nearly spherical outward propagating flames passing through homogeneous and stratified mixtures were investigated both experimentally and numerically. Constant volume chamber experiments were observed from Schlieren image frames captured by a high speed camera and processed by a semi-autonomous in-house developed parallelized object oriented code to robustly track the surface flame front and the location of the bubble to determine the relevant combustion characteristics.

The flame surface speeds of the experiment were compared with both simulation and past work in the literature. The flame surface speed enhancements were noted in the stratified rich to lean cases assuming the flame surface structure remained smooth, in agreement with past literature. Qualitative agreement was achieved with the numerical results, agreeing with and supporting the literature sited mechanisms of both thermal and hydrogen preferential diffusion. Emission data was measured for the cases considered and the stratified flames exhibited reduced unburned hydrocarbons and increased CO production and reduced NO_x production. Additionally, carbon monoxide production increased with the stratification as well as a decrease in CO_2 production.

Unexpectedly, the flame surface exhibited cellular structures during the transition region which complicates the findings. Even accounting for the limited effects of the soap bubble, this work has found that both of the stratified rich to lean or stratified lean to rich flames are likely unstable flames due to the thermo-diffusive instabilities present surrounding the flame during the transition. Upon compare three dimensional combustion simulations, a set of cellular structures appeared in the flame front surface agreeing with the experimental determination. Although the flame speed enhancements noted in this work could not be fully decoupled from the cellular structures noted in the images of the flame surface, this stratified enhancement effect appears to be real.

The lean limit was explored in a stratified setup and appeared to be extended. However further local equivalence ratio determination in the region is needed to fully confirm the extent of severely lean burning.

8.2 Future Directions

As there are many applications as well as in the pursuit of greater fundamental knowledge of the relevant physical processes inherent in current and future power production technology, stratified flame combustion research is needed. Both experimentally and numerically, further investigations of the stratification effect are needed to continue to decouple from fluid dynamics and quantify the effects stratification has on laminar flames.

Numerical directions

While past and numerical work supports the existence of inherent instabilities within the stratified flames, further investigation is necessary to completely tease out if cellularity is inherent in the stratification or only a product of various experimental setups. More detailed modeling of the bubble is necessary to control for the fluid dynamic instabilities present as the flame approaches and burns through this region. A more extensive numerical study able to accurately model the three dimensional instability effects is needed to explore the extent the instabilities resulting from stratified flames persist. In particular, additional comparisons with planar and spherical geometries with stratification layers of different size and different stretch rates may assist in determining the limits of the instabilities observed in the experiment. Furthermore, additional work is required to investigate the effects of stratification on turbulence seen in numerous experiments and real world application.

Possible Experimental Directions

Experimentally, modifications to the experimental setup will assist in isolating key effects. With testing different sized spherical stratified test setups, the effect of stretch on the stratification layer can be further studied. Additional sizes of optical view area relative to the bubble size will assist in better tracking of the flame surface into the lean limit. Additional diagnostics to determine the local equivalence ratio including preferential diffusion of hydrogen species is needed to confirm extension of the lean flammability limit.

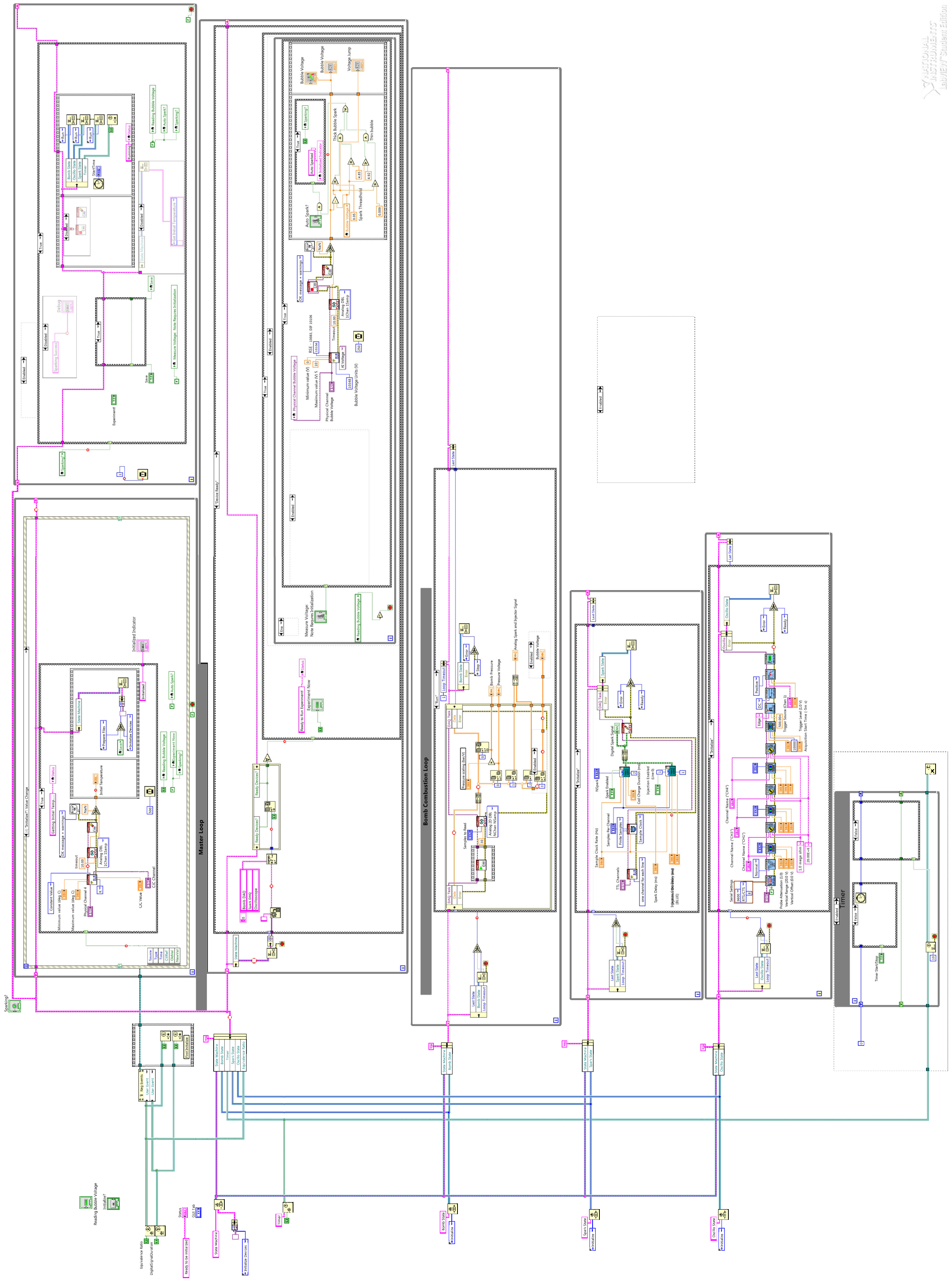
Appendix A

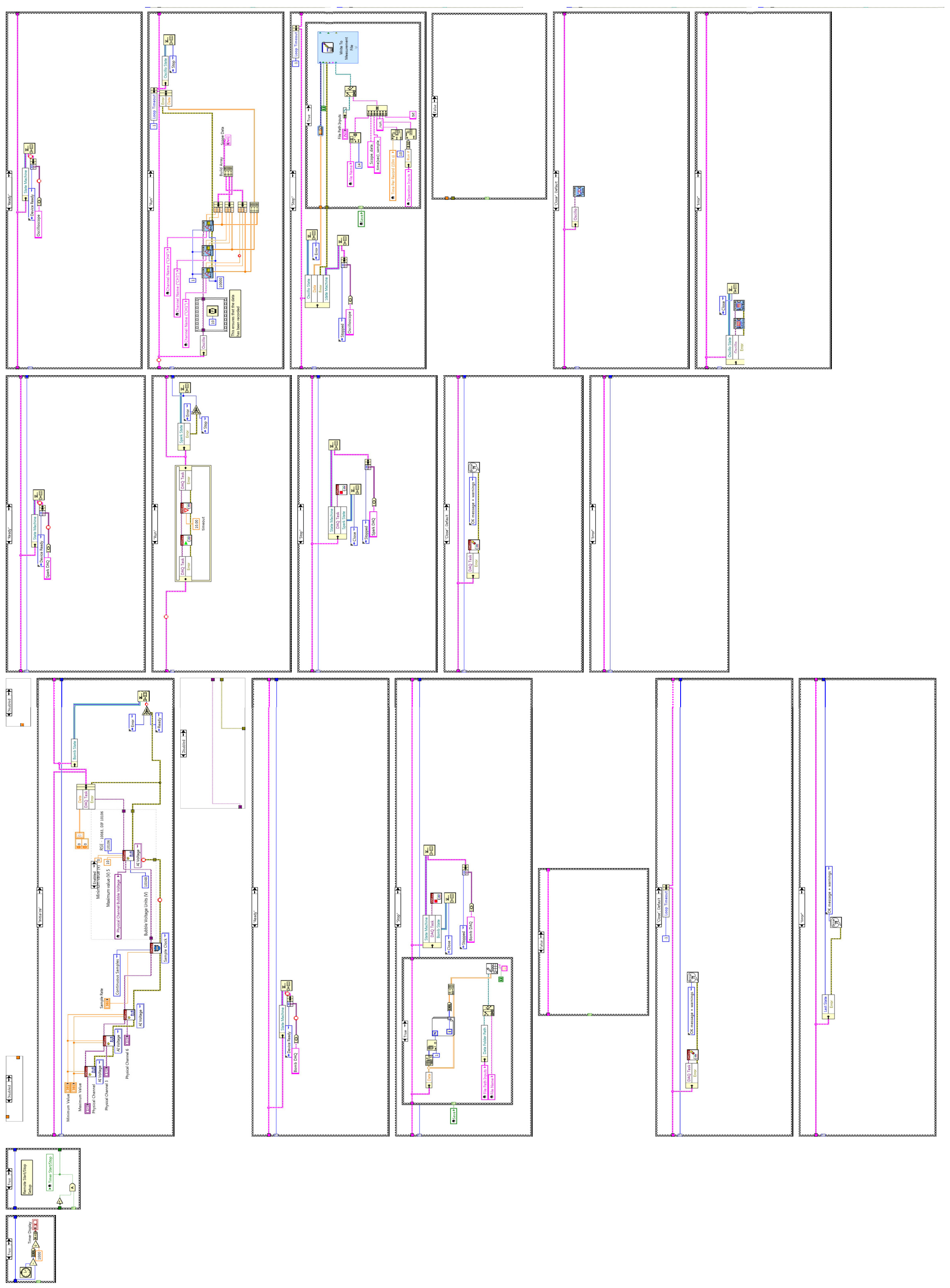
Appendix

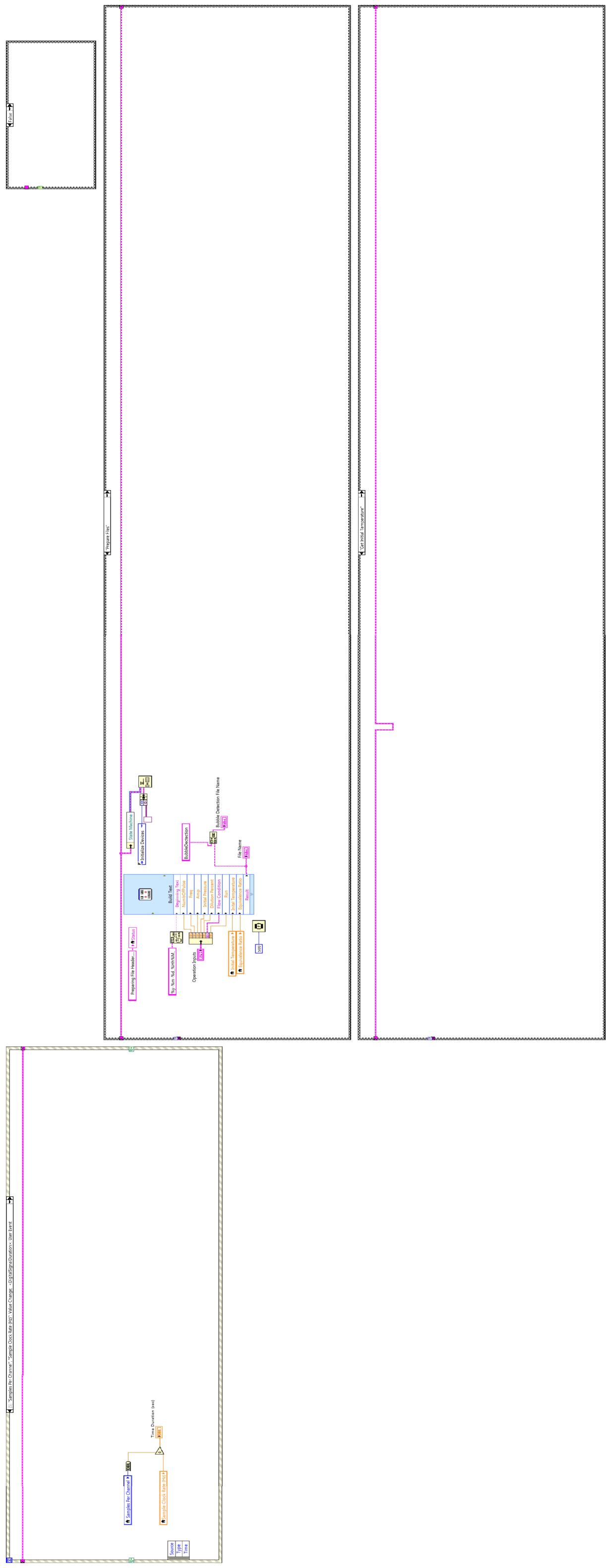
A.1 LabView Block Diagram

The custom LabVIEW program utilized in the data aquisition system is presented in this appendix. The following pages outline the block diagram used.

Block Diagrams for DAQ


Master Loop
Bomb Combustion Loop
Timer





A.2 Post Processing Code

The custom post proessing code to edge track the flame surface and bubble fronts is presented in this appendix.

Main Driving Code

```
function ProcessSchlieren_Bubble(folderPath, flameCutoffThreshold, bubbleCutoffThresh, ...
    SolidBodyThreshold, autoFlameLocMinBrtThresh, maxFlameSpeedPhysical_cms, ...
    initialChambPressure_psig, phiInit, phiPost, isBubble, tank1Tup, tank1Date, ...
    tank2Tup, tank2Date, frameBubPopped, isWet)
% Schlieren Movie Post Process Code by Charles Scudiere
%
% Main paramters to change: (make sure to adjust Tambient and Tadiabatic
% for the specific conditions tested at (In Filename and PREMIX))
% Homogenous cases => Target average temperature
% Bubble cases => Pin center tracks as best as possible
%
% folderPath - location of movie and pressure data files
%
% flameCutoffThreshold - lower if bubble showing in flame tracker
% 0-255 (use to be between 0 and 1), cutoff for black/white framing.
% Adjust to better track flame front, 30-80 reasonable
% Increase to see lighter points, decrease to see darker points
%
% bubbleCutoffThresh - identical to flameCutoffThreshold, but for the bubble
%
% SolidBodyThreshold - Lower values (Darker) from first frame are
% converted to white, decrease to remove non-body whited parts in
% ModifiedFrames photos
%
% autoFlameBrightnessThresh - (Obsolete) - Local Min determination
% Threshold to check light intensity local mins to avoid
% catching the bubble too as in the raw darkness cases. Lower if
% catching bubble, raise if not tracking flame.
%
% maxFlameSpeedPhysical_cms - [cm/s] Maximum flame speed expected, to
% eliminate unphysical jumps to bubble.
%
% initialChambPressure_psig - [PSIG] Initial pressure in Chamber before spark
% Defaults to 0.5 PSIG
%
% phiInit & phiPost - Equivalence ratio before and after bubble or early and
% late burn properties. Defaults to -1, plotting homogenous values
%
% isBubble - Whether or not there is a bubble in the video
%
%
% tank1Tup/tank2Tup - tank 3 pressure tuple for equivalence ratio determination
%
% tank1Date/tank2Date - Date tank made, for serialing tank mixtures
%
%
% frameBubPopped - frame bubble pop influences pin center row (0 for no bubble)
%
% isWet - Logical of whether chamber is wet
%
```

```
%
% Main Structures in Code:
% PSUsrInput - User input related vars
% PSConditions - Conditions that don't change on a particular run,
% but includes vars passed from function call
% PSFlags - Flags for debug, plot display, etc.
% PSRunData - Main Data is storred here
% PSScopeData - Scope and Pressure Data is Storred here
% PSLoopingParam - Looping parameters that may change on each of
% the main loop iterations

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FLAGS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if true
    %% Running Flags
    PSFlags.isRunLoop=true;
        % Whether to run to loop through videos or grab previus session vars
        % WARNING overwrites user input data with previous values used in loop

    PSFlags.debug_skipUsrInput=true;
        % Whether or not to skip code getting new user input
        % Grabbs previous user input from previously saved matlabsession file
        % for faster loop parameter processing

    PSFlags.isJustGetUsrInput=false;
        % Whether to just get the user input and then exit
        % NOTE: isRunLoop flag must be true to gather correctly

    PSFlags.isTrackCirCenter=~isBubble;
        % Whether or not to use circle center to move tracking row
    PSFlags.isUseTrackedCirCenter=false; %false;
        % Whether or not to use tracked circle center for flame radius with
        % merging with pin center

    PSFlags.isForceBubble=false;
        % Force run to have bubble when running loop is selected, in cases where
        % user input was taken with a bubble but errored
    PSFlags.isRedoPressureNSpark=false;
        % Force rereading DAQ txt files, scope and pulse data: plotting, etc.
    PSFlags.isForceUsrInputBubbleDia=false;
        % Whether or not to grab user input for bubble diameter

    PSFlags.isFilterRadius=false;
        % false sets PSPostProcData.r_cm_interp to splined value
        % true sets PSPostProcData.r_cm_interp to filtered splined value

    %% Display Flags
    PSFlags.dispLoopingPlot=true;
        % Displays looping plot
    PSFlags.dispKeyPlots=false;
        % Displays post-processing plots

    PSFlags.plotBrightnessHistogram=false;
        % Plots histogram next to frame image plot in a sub plot
    PSFlags.debug_autoScaleAxis=false;
        % Whether or not to plot on a scale determined to show ~all the data or
        % a specified axes plotting...disable for comparison between plots

    PSFlags.isPrintFlameCM=true;
```

```
        % Whether or not to output the flame front location [cm] to the screen

    %% Presentation/Photo Saving Flags
    PSFlags.saveFrameStills=false;
        % False better default, speeds up code.
        % For Presentations, saves each video frame separately.
        % Wheter or not to make and save folder of avi frame jpg images
    PSFlags.saveImgFrameOnly=false;
        % Whether or not to only plot the image and not plot the row tracked data
    PSFlags.isZoomRightSide=false;
        % Whether to automatically zoom into region of interest
        % from pin center to little to the right of the flame location
    PSFlags.isZoomToTrackRow=true;
        % If zoom based on maximum track row

    PSFlags.isPlotCirOnly=false;
        % Will not plot schlieren, only detected circle center of flame

    PSFlags.isAddScale=false;
        % Whether or not to add a scale to frame stills in bottom right corner

    %% Speed up Flags
    PSFlags.suppressDebugOutput=true;
        % True is faster. Less output for error checking, etc.
    PSFlags.useMatlabVideoReader=false;
        % False is faster, uses a modified matlab structure (same format) that only uses
        % the read functionality for speed up and no frame conversion.
        % Works for Greyscale cases!
    PSFlags.useMatlabVideoWriter=true;
        % True is faster. Exploring speedups - writing images for ffmpeg later.
        % Using EPSC is better quality but much much slower
    PSFlags.writeVideoAtEndOfLoop=true;
        % True is better. Whether to write frame at each frame by frame (false),
        % or all at once at the end of looping through the frames (true).
        % True has less wear on hard drive but marginal speed improvement.

    PSFlags.reduceLoopFeatures=true;
        % True is faster.
        % Speed Up by removing legend and other time consuming items on loop ploting

    %% Saving work Flags (Costs time but records values)
    PSFlags.saveOutVideo=false;
        % Whether or not to save output video, so speed up code and get input correct
    isSortNameVideo=PSFlags.saveOutVideo;
        % Whether or not to copy output video to a name that is more descriptive

    PSFlags.savePostPlots=false;
        % Whether or not to save post-processing plots

    PSFlags.saveOrigStills=false;
        % False speeds up code, do once. For saving original frames as jpg/png in folder.
        % False better default, speeds up code.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
%% END FLAGS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
funcStartTime=cputime;
    % Start timer for timing output in looping
```

```
%% Suppress Warnings
fprintf('\n\nStarting Post-Processing on Video in folder: %s\n', folderPath)

if PSFlags.suppressDebugOutput
   warning('off');
   fprintf('\tSuppressing Most Debug Output and warnings\n')
else
   warning('on');
   warning('off', 'Images:initSize:adjustingMag');
       % Suppress warning of "Warning: Image is too big to fit on screen"
   warning('off', 'MATLAB:subscripting:noSubscriptsSpecified');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Load Known/Passed Conditions, File Information, etc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PSConditions.folderPath = folderPath;
PSConditions.flameCutoffThreshold = flameCutoffThreshold;
PSConditions.bubbleCutoffThresh = bubbleCutoffThresh; % Too low gives false bubble pops...
PSConditions.SolidBodyThreshold = SolidBodyThreshold;
PSConditions.autoFlameLocMinBrtThresh = autoFlameLocMinBrtThresh;
PSConditions.maxFlameSpeedPhysical_cms = maxFlameSpeedPhysical_cms;
PSConditions.initialChambPressure_psig = initialChambPressure_psig;
PSConditions.isBubble=isBubble;
PSConditions.frameBubPopped = frameBubPopped;

if phiInit ~= phiPost
    % Calculate Initial spark region Tank Phi
    phiInit_calc = getPSTankEqR(tank1Tup, tank1Date);
    fprintf('\tInit: %f Given vs. %f Calculated from tank made %s\n', ...
        phiInit, phiInit_calc, tank1Date)

    phiPost_calc = getPSTankEqR(tank2Tup, tank2Date);
    fprintf('\tPost: %f Given vs. %f Calculated from tank made %s\n', ...
        phiPost, phiPost_calc, tank2Date)
else
    % Calculate Initial spark region Tank Phi
    phiInit_calc = getPSTankEqR(tank1Tup, tank1Date);
    fprintf('\tInit: %f Given vs. %f Calculated from tank made %s\n', ...
        phiInit, phiInit_calc, tank1Date)
    phiPost_calc=phiInit_calc; % Same when no bubble.
end
PSConditions.phiInit = phiInit_calc; %phiInit;
PSConditions.phiPost = phiPost_calc; %phiPost;

printPSEquivalenceRatio(phiInit, phiPost, PSConditions, true)
    % Checks Equivalence Ratio to ensure in correct bin, and prints it

if PSConditions.phiInit == PSConditions.phiPost
        PSConditions.isHomogenous=true;
                % Whether or not to plot simulation curves for Homogenous Runs: .65, 1, 1.4
    PSConditions.isRichLeanStrat=false;
elseif PSConditions.phiInit > PSConditions.phiPost
        PSConditions.isHomogenous=false;

    PSConditions.isRichLeanStrat=true;
                % Whether or not to plot simulation curves for stratified from 1.4->.65
elseif PSConditions.phiInit < PSConditions.phiPost
```

```
    % If Lean to Rich Stratified!
        PSConditions.isHomogenous=false;

    PSConditions.isRichLeanStrat=false;
                % Whether or not to plot simulation curves for stratified from .65->1.4
        fprintf('\tNote: No current graphs for Lean to Rich Implemented\n')
end
PSConditions.isWet=isWet;

% Auto detect video filename using system ls command:
try
    aviFilenameList=ls(strcat(PSConditions.folderPath, filesep,'*.avi'));
    aviFilenames=split(aviFilenameList, PSConditions.folderPath);
    [~, PSConditions.VideoName, Video_ext ] = fileparts(aviFilenames{2});
    if ~strcmp(Video_ext(1:4),'.avi')
        fprintf('Error detecting avi file\n')
    end
catch e
    fprintf('Error determining foldername with %s.\n', PSConditions.folderPath)
    PSConditions.VideoName = PSConditions.folderPath(end-14:end);
    Video_ext = '.avi';
    fprintf('Minor Error in detmining folderPathLoc: %s Occured in post-Processing %s\n',...
        e.identifier, PSConditions.VideoName)
    fprintf('\t%s\n', e.message)

    if ~exist(fullfile( strcat(PSConditions.folderPath, filesep, PSConditions.VideoName, ...
            '.avi')), 'file')
        fprintf('MAJOR ERROR: avi file %s does not exist...exiting.\n', ...
            [PSConditions.VideoName, '.avi'])
        return
    else
        fprintf('Recovered setting VideoName (%s) from folderpath (%s)\n', ...
            PSConditions.VideoName, PSConditions.folderPath)
    end
end
if length(PSConditions.VideoName)>=18 && ...
        strcmp(PSConditions.VideoName(1:18),'Flame_Propagation_')
    % If just post-processing and only video is the Flame_Prop video and the matlab file...
    PSConditions.VideoName = PSConditions.VideoName(19:end);
end

PSConditions.rawVideofilename = ...
    char(strcat(PSConditions.VideoName,Video_ext)); % video file name
    % Matlab likes character arrays over strings...

% See if previous matlab saving format exists
PSConditions.prevRunMatlabFN= ...
    fullfile(strcat(PSConditions.folderPath, filesep,'MatlabSessionVars.mat'));
PSConditions.specificRunMatlabFN = ...
    fullfile(strcat(PSConditions.folderPath, filesep,'MatlabSessionVars_', ...
    PSConditions.VideoName,'.mat'));

PSFlags.isPrevRunMatlab=exist( PSConditions.prevRunMatlabFN, 'file') == 2;
PSFlags.isSpecificRunMatlab=exist( PSConditions.specificRunMatlabFN, 'file') ==2;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Check User Input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if PSConditions.isBubble && ~PSConditions.frameBubPopped
```

```
    % If bubble and bub pop frame is 0...can't happen!
    fprintf('\n\tWARNING in reading %s: Bubble but no valid pop frame input\n', ...
        folderPath)
    pause(3)

elseif ~isBubble && frameBubPopped
    % If no bubble and pop frame isn't 0...can't happen!
    fprintf('\n\tWARNING in reading %s: No bubble but input frame when it popped!\n', ...
        folderPath)
    pause(60)
end
if PSConditions.isBubble && ~PSConditions.isWet
    % If not given wet condition but obviously should be wet...
    fprintf('\n\tWARNING in reading %s: Marked as NOT WET, but bubble exits?!\n', ...
        folderPath)
    pause(60)
end

%% End Check User Input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Video Radius Gathering and Post-Processing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

try
    % Use try-catch to ensure parallel goes forward
    %% Load Previous Bub/Struct
    if ~PSFlags.isRunLoop || PSFlags.debug_skipUsrInput
        [PSConditions.isBubble, PSFlags.isPrevRunMatlabStruct] = ...
            grabPrevBubNMatSaveType(isBubble, PSConditions, PSFlags);
    else
        % Else running a normal run to grab usr input and run the loop
        % (Or exit early with the just grab user input flag)
        PSFlags.isPrevRunMatlabStruct=false;
        % Must be set when grabbing user input to check and see if in structure format
        % Mark as false as running loop without grabbing previous input
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Post-Process Pressure/Scope/FPS Data
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [PSScopeData, PSConditions] = getPSPresScopeFpsFileData(PSConditions, PSFlags);

    PSConditions.dBubble_assumed_cm=2.9; % [cm] Diameter of assumed bubble

    % Check bubble size
    if PSFlags.isRunLoop
        if ~PSFlags.suppressDebugOutput
            fprintf('\tFPS=%d, assuming bubble size of %.3f cm\n',PSConditions.fps, ...
                PSConditions.dBubble_assumed_cm);
            if PSConditions.isBubble
                fprintf('\tCheck further below for selected bubble size for validation.\n')
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Get User Input
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    PSUsrInput = getPSUsrInput(PSConditions, PSFlags);
    % return % For debugging bubble brightness, etc. from user input
    % Save User Input Data for later runs
    if PSFlags.isJustGetUsrInput
        %save(PSConditions.prevRunMatlabFN)
        save(PSConditions.specificRunMatlabFN)
        fprintf('Got User Input Saved to Disk, now exiting\n')
        return % exit just exits matlab...
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Loop through frames
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if PSFlags.isRunLoop
        fprintf('\n\tRunning loop through video frames:\n')

        PSLoopingParam = initPSLoopingParam(PSConditions, PSUsrInput, PSFlags);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% Read RAW Video for all frames in 4-D array of size(height,width, 3, nFrames)

        if ~PSFlags.useMatlabVideoReader
            if PSUsrInput.isRightDark
                PSAllFrames = read(PSLoopingParam.HSCmovObj);
                % Try Reading only (without format converting), using own version of read
            else
                % If dark is on the left hand size, flip read video frames to RHS
                % User input is not flipped and horizontal values are flipped along axis.
                PSAllFrames = flip(read(PSLoopingParam.HSCmovObj), 2);

                % Flip user input if needed
                if PSUsrInput.pixEdgeOfRIO < PSUsrInput.pinsCenter_x
                    % If user input hasn't been flipped yet
                    PSUsrInput.pinsCenter_x = size(PSAllFrames(:,:,1), 1)/2 +...
                        (size(PSAllFrames(:,:,1), 1)/2 - PSUsrInput.pinsCenter_x);
                    PSUsrInput.pixEdgeOfRIO = size(PSAllFrames(:,:,1), 1)/2 +...
                        (size(PSAllFrames(:,:,1), 1)/2 - PSUsrInput.pixEdgeOfRIO);

                    if PSConditions.isBubble
                        % Update the values not already accounted for
                        pixel_bubble_right_avg = ...
                          size(PSAllFrames(:,:,1), 1)/2 + (size(PSAllFrames(:,:,1), 1)/2 ...
                        - PSUsrInput.pixel_bubble_left_avg);
                            %Shouldn't this be left average and flip to right?
                        pixel_bubble_left_avg = ...
                          size(PSAllFrames(:,:,1), 1)/2 + (size(PSAllFrames(:,:,1), 1)/2 ...
                            - PSUsrInput.pixel_bubble_right_avg);

                        % Now overwrite value, since flipping
                        PSUsrInput.pixel_bubble_right_avg=pixel_bubble_right_avg;
                        PSUsrInput.pixel_bubble_left_avg=pixel_bubble_left_avg;

                        % Fix bubble radius
                        PSUsrInput.rBubble_cm = ...
                            (PSUsrInput.pixel_bubble_right_avg - ...
                                PSUsrInput.pinsCenter_x)/PSUsrInput.pixel2cm;
                    end
                end
            end
        end
```

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

%% Determine Video Conditions from PSLoopingParam HSCmov file
PSConditions.totaltime=PSLoopingParam.HSCmovObj.duration;
    %seconds, duration of the file, NOT the physical time
PSConditions.numFrames = ...
    PSLoopingParam.HSCmovObj.duration*PSLoopingParam.HSCmovObj.frameRate;
PSConditions.truetotaltime=PSConditions.numFrames/PSConditions.fps;
    %true physical time in which event took place
PSConditions.raw2TrueTimeRatio=...
    PSConditions.truetotaltime/PSLoopingParam.HSCmovObj.duration;

%% Initialize flame tracking RunData
% Allocate to numFrames, will crop later to PSLoopingParam.frameVidEnd
PSRunData.dumbflameloc=zeros(PSConditions.numFrames,1);
PSRunData.autoflameloc_cirCen=zeros(PSConditions.numFrames,1);
PSRunData.autoflameloc_pinCen=zeros(PSConditions.numFrames,1);

PSRunData.xCirCenter=zeros(PSConditions.numFrames,1);
PSRunData.yCirCenter=zeros(PSConditions.numFrames,1);
PSRunData.CirRad=zeros(PSConditions.numFrames,1);

PSRunData.circleCenterLeadRow = zeros(PSConditions.numFrames,1);

%% Initialize ROW flame tracking RowData
PSRunData.j_frameCol_pinCenter = int64(PSUsrInput.pinsCenter_y);
circleCenterLead=int64(PSUsrInput.pinsCenter_y);
    % First track at pin center, moves as needed

%% Initialize the initial 2D Array
% parfor
for iRow=PSRunData.j_frameCol_pinCenter-PSLoopingParam.numRowPrior:...
        PSRunData.j_frameCol_pinCenter+PSLoopingParam.numRowPost
    PSRunData.PSRows(iRow) = initPSRowData(iRow, PSConditions, PSUsrInput, ...
        PSLoopingParam, PSAllFrames(:,:,:,1));
        % Blanks are ok, the blanks are taking only 85% of one full struct ~10 kB
    if iRow == PSRunData.j_frameCol_pinCenter
        PSRunData.PSRows(iRow).isLead = true; % Start with lead as pin center
    end
end
PSRowData = PSRunData.PSRows(PSRunData.j_frameCol_pinCenter);

%% Create an out avi video file if saving video
if PSFlags.saveOutVideo
    if PSFlags.useMatlabVideoWriter
        outputVidflnm =['Flame_Propagation_',PSConditions.rawVideofilename];
            % Name of file to be saved
        outputVideoPath=fullfile(PSConditions.folderPath, outputVidflnm);
        outputVideoObj=VideoWriter(outputVideoPath); %FileFormat
        outputVideoObj.FrameRate=...
            PSConditions.numFrames/PSConditions.totaltime*0.125;
            % increase to make faster
        outputVideoObj.Quality=100;
        open(outputVideoObj);


        PSAllOutputFrames=...
            struct('cdata', uint8(zeros(1024,1024, 3)), 'colormap', []);
            % Initialize initial structure array used to save video frames,
            % but will adjust as necessary
    else
        if ~exist(fullfile(strcat(PSConditions.folderPath, filesep,'MovieFrames')),...
                'dir')
            mkdir(fullfile(strcat(PSConditions.folderPath, filesep,'MovieFrames')));
        end
        outputVidflnm =['Flame_Propagation_',PSConditions.rawVideofilename];
            % Prefix of Name to be saved
        outputVideoPath=...
            fullfile(PSConditions.folderPath, filesep,'MovieFrames', outputVidflnm);
    end
end

%% Begin video looping
PSLoopingStartTime=cputime;
frameLoopingFigHand = figure('Visible', 'On', 'Position', ...
    [900, 245, 1120, 840]);
    % 'Position', [left bottom width height]
    % Make looping frame larger for higher res video
if ~PSFlags.dispLoopingPlot
   frameLoopingFigHand.Visible='off';
end
dispPercent=5;
printUpdateFreq=int8(PSConditions.numFrames*dispPercent/100);
    % int8(PSConditions.numFrames/10); % Print about every X percent

while PSLoopingParam.iCurrentFrame <= PSLoopingParam.frameVidEnd
    %% Output Current Progress/Status

    if ~mod(PSLoopingParam.iCurrentFrame-1,printUpdateFreq)
        % Print compltion rate every *th step
        if PSLoopingParam.iCurrentFrame~=1
            PSLoopingElapsedTime=cputime-PSLoopingStartTime;
            dateTime=clock;

            pinLeadPercComp = 100*...
                (PSRunData.leadPSRow.autoflameloc(PSLoopingParam.iCurrentFrame-1)...
                - PSUsrInput.pinsCenter_x)/...
                    (PSUsrInput.pixEdgeOfRIO - PSUsrInput.pinsCenter_x);

            fprintf(strcat('@ %d:%02d, %.1f%% complete with %.2f %% of input', ...
                'read from video %s.avi, at ', ...
                dateTime(4), dateTime(5), pinLeadPercComp, ...
                100*PSLoopingParam.iCurrentFrame/PSConditions.numFrames, ...
                PSConditions.VideoName);
            fprintf('~%.2f frames/sec. Elapsed time: %.0f sec\n', ...
                (PSLoopingParam.iCurrentFrame-1)/PSLoopingElapsedTime, ...
                PSLoopingElapsedTime)
        end
    end

    %% Read next frame
    if PSFlags.useMatlabVideoReader
        PSLoopingParam.frame_orig = readFrame(PSLoopingParam.HSCmovObj);
            %read(mov,i); %reads the specified frame
    else
        PSLoopingParam.frame_orig = ...
            PSAllFrames(:,:,:, PSLoopingParam.iCurrentFrame);
            % reads the specified frame
    end
    PSLoopingParam.frame = PSLoopingParam.frame_orig;
        % Keep for later plotting against original frame

    if PSLoopingParam.iCurrentFrame==1
        PSLoopingParam.firstFrame=PSLoopingParam.frame;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %% Check to see if flame has expanded and need to add more rows
    if PSLoopingParam.isFlameStart

        maxGrowSize=10; %50;%20;
        testSize=min( int64(min(0.25*PSLoopingParam.numRowPrior, ...
            max(size(PSLoopingParam.frame,2)-PSRunData.j_frameCol_pinCenter, 0)...
            )), maxGrowSize);
            % Test for this number of rows to see if one of them has ignited to add
        addSize=min( int64(min(0.25*PSLoopingParam.numRowPrior, ...
            max(size(PSLoopingParam.frame,2)-PSRunData.j_frameCol_pinCenter, 0)...
            )), maxGrowSize);
            % Number of rows to start tracking
            % Test to see if need to add more rows Prior
        addPrior=false;
        if PSLoopingParam.numRowPrior <= PSLoopingParam.numRowPriorMAX
            % If can add more points prior the center (enough points before)
            for iRow=PSRunData.j_frameCol_pinCenter-PSLoopingParam.numRowPrior...
                    :PSRunData.j_frameCol_pinCenter-PSLoopingParam.numRowPrior+testSize
                if PSRunData.PSRows(iRow).isFlameStart
                    addPrior=true;
                end
            end
            if addPrior
                % If need to add more points prior the center column, add them
                startRange=max(PSRunData.j_frameCol_pinCenter-...
                    PSLoopingParam.numRowPrior-addSize, 1);
                endRange=...
                    (PSRunData.j_frameCol_pinCenter-PSLoopingParam.numRowPrior-1);
                currAddSize=length(startRange:endRange);
                for iRow=startRange:endRange
                    PSRunData.PSRows(iRow) = ...
                        initPSRowData(iRow, PSConditions, PSUsrInput, ...
                        PSLoopingParam, PSAllFrames(:,:,:,1));
                end
                PSLoopingParam.numRowPrior=PSLoopingParam.numRowPrior+currAddSize;
            end
        end
        % Test to see if need to add more rows Post
        addPost=false;
        if PSLoopingParam.numRowPost <= PSLoopingParam.numRowPostMAX
            % If can add more points post the center
            for iRow=PSRunData.j_frameCol_pinCenter+...
                    PSLoopingParam.numRowPost-testSize:...
                        PSRunData.j_frameCol_pinCenter+PSLoopingParam.numRowPost
                if PSRunData.PSRows(iRow).isFlameStart
                    addPost=true;
                end
            end
            if addPost
                % If need to add more points post the center column, add them
                startRange=(PSRunData.j_frameCol_pinCenter+...
                    PSLoopingParam.numRowPost+1);
                endRange=min(PSRunData.j_frameCol_pinCenter+...
                   PSLoopingParam.numRowPost+testSize, ...
                   size(PSLoopingParam.frame_orig, 1));
                currAddSize=length(startRange:endRange);
                for iRow=startRange:endRange
                   PSRunData.PSRows(iRow) = initPSRowData(iRow, PSConditions, ...
                      PSUsrInput, PSLoopingParam, PSAllFrames(:,:,:,1));
                end
                PSLoopingParam.numRowPost = PSLoopingParam.numRowPost+currAddSize;
             end
          end
       end

       %% Find location of flame, cir. cen, etc.
       % j_frameCol=int16(PSUsrInput.pinsCenter_y); % Which row we are looking at
       PSLoopingParam.leadJ_FrameCol = int64(PSUsrInput.pinsCenter_y);

       PSRowData = getPSRowFlameLoc(PSLoopingParam.leadJ_FrameCol, PSRowData, ...
          PSRunData, PSConditions, PSUsrInput, PSLoopingParam, PSFlags);

       % Save Lead Row to Run Data
       PSRunData.leadPSRow = PSRowData;

       % Row Plotting Parameters
       PSLoopingParam.autoPickedPixelslocFull = PSRowData.autoPickedPixelslocFull;
       PSLoopingParam.testVectEndIdx = PSRowData.testVectEndIdx;
       PSLoopingParam.timesPrevJumpedSoModAdv = ...
          PSLoopingParam.timesPrevJumpedSoModAdv+ PSRowData.isJumpedSoModAdv;
          % Add 1 if held back with modified advancement from a previous jump

       % Detect if flame just started and set appropriate parameters
       if ~PSLoopingParam.isFlameStart && (PSRowData.flameKernalDevelFrameNumb ~=...
             PSLoopingParam.flameKernalDevelFrameNumb)
          % If detected flame starting, reset PSLoopingParam Frame Number Limit.
          % Should happen once as setting equality in getFlameLocation Function!
          PSLoopingParam.flameKernalDevelFrameNumb=...
             PSRowData.flameKernalDevelFrameNumb;

       elseif ~PSLoopingParam.autoHitEdge && (PSRowData.frameVidEnd ~= ...
             PSLoopingParam.frameVidEnd)
          % If detected flame ending, reset PSLoopingParam VidEnd Limit.
          % Should happen once as setting equality in getFlameLocation Function!
          PSLoopingParam.frameVidEnd = PSRowData.frameVidEnd;
       end
       PSLoopingParam.isFlameStart = PSRowData.isFlameStart;
       PSLoopingParam.hitEdge = PSRowData.hitEdge;
       PSLoopingParam.autoHitEdge = PSRowData.autoHitEdge;

       % Update Rows to Run Data
       PSRunData.leadPSRow = PSRowData;
       leadPSRow=PSRunData.leadPSRow.frameCol;
       if PSLoopingParam.isFlameStart
          % Previous value of autoflameloc is used.
          % Since skipping up until pin center detects flame,
          % prior values set when detect if flame just started
          numbRowsBubblePopped=0;
          % parfor % if not using spmd
          for iRow=PSRunData.j_frameCol_pinCenter-PSLoopingParam.numRowPrior:...
                PSRunData.j_frameCol_pinCenter+PSLoopingParam.numRowPost
             if ~PSRunData.PSRows(iRow).isLead
```

```
        PSRunData.PSRows(iRow) = getPSRowFlameLoc(iRow, ...
            PSRunData.PSRows(iRow), PSRunData, PSConditions, ...
            PSUsrInput, PSLoopingParam, PSFlags);
        if PSRunData.PSRows(iRow).autoflameloc(PSLoopingParam.iCurrentFrame) ...
                > PSRunData.PSRows(leadPSRow).autoflameloc(PSLoopingParam.iCurrentFrame)
            % Find lead row in this set. Only need to test here as starting with lead flame.
            leadPSRow=iRow;
        end
    else
        PSRunData.PSRows(iRow) = PSRunData.leadPSRow;
    end
    if ~PSLoopingParam.BubblePopedStartedFrame &&...
            PSRunData.PSRows(iRow).BubblePopedFrame
        numbRowsBubblePopped=numbRowsBubblePopped+1;
    end
end
if ~PSLoopingParam.BubblePopedStartedFrame ...
        && (numbRowsBubblePopped-PSLoopingParam.prevNumbRowsBubblePopped >...
        PSLoopingParam.numRowBubblePopThresh)
    % Look at rate of row bubble pops to detect when bubble actually popped.
    % If several rows indicate bubble has popped, mark it!!
    PSLoopingParam.BubblePopedStartedFrame = PSLoopingParam.iCurrentFrame;
    fprintf(['\tAt frame %d VS %d passed for %s.avi.',...
        'Speculatively Detected bubble popped (Lost bubble tracking)!\n'], ...
        PSLoopingParam.BubblePopedStartedFrame, ...
        PSConditions.frameBubPopped, PSConditions.VideoName)
end
PSLoopingParam.prevNumbRowsBubblePopped = numbRowsBubblePopped;

% Determine Aparent Center for frame segment, uses prior and post points
[PSRunData.xCirCenter(PSLoopingParam.iCurrentFrame), ...
    PSRunData.yCirCenter(PSLoopingParam.iCurrentFrame), ...
    PSRunData.CirRad(PSLoopingParam.iCurrentFrame)] ...
    = getPSAparentFlameCenter(PSRunData, PSUsrInput, PSLoopingParam);

% Find Lead Flame Row!
possibleNewLeadRow=int64(PSRunData.yCirCenter(PSLoopingParam.iCurrentFrame));
if possibleNewLeadRow<=PSRunData.j_frameCol_pinCenter+PSLoopingParam.numRowPost ... % If in
     bound set
        && possibleNewLeadRow>=PSRunData.j_frameCol_pinCenter-PSLoopingParam.numRowPrior
    circleCenterLead=possibleNewLeadRow;
end

% Determine and Update Aparent Radius for stretch rate calc
for iRow=PSRunData.j_frameCol_pinCenter-PSLoopingParam.numRowPrior:...
        PSRunData.j_frameCol_pinCenter+PSLoopingParam.numRowPost
    PSRunData.PSRows(iRow) = getPSRowFlameStretchRad(...
        PSRunData.PSRows(iRow), PSRunData, PSLoopingParam);

    % Determine deviation from circle center and radius
    PSRunData.rowRadDevs(iRow)=...
        (PSRunData.CirRad(PSLoopingParam.iCurrentFrame)- ...
        PSRunData.PSRows(iRow).StretchRad(PSLoopingParam.iCurrentFrame));
    if abs(PSRunData.rowRadDevs(iRow))>=PSLoopingParam.flameRadDevTol ...
            && PSRunData.PSRows(iRow).isFlameStart &&...
            ~PSRunData.PSRows(iRow).autoHitEdge
        PSRunData.PSRows(iRow).isRowSuspectAtFrame(PSLoopingParam.iCurrentFrame) = true;
    end
end
rowIterPts=PSRunData.j_frameCol_pinCenter-PSLoopingParam.numRowPrior:...
```

```
        PSRunData.j_frameCol_pinCenter+PSLoopingParam.numRowPost;
    PSRunData.TotalDev(PSLoopingParam.iCurrentFrame) = ...
        (sum(PSRunData.rowRadDevs(rowIterPts).^2))^0.5;

    PSRowData = getPSRowFlameStretchRad(PSRowData, PSRunData, PSLoopingParam);
end
% Update Rows to Run Data
PSRunData.leadPSRow = PSRowData;

if ~PSLoopingParam.autoHitEdge ...
        && (isPSRowExist(PSRunData, PSLoopingParam.trackingRow) && ...
        ~PSRunData.PSRows(PSLoopingParam.trackingRow).autoHitEdge)
    % Save tracking row as tracked lead row since following flame lead here...
    PSRunData.circleCenterLeadRow(PSLoopingParam.iCurrentFrame) = ...
        circleCenterLead;
    if PSFlags.isTrackCirCenter
        % Move tracking row if tracking circle center flag set
        PSLoopingParam.trackingRow = circleCenterLead;
    end
elseif ~PSFlags.isTrackCirCenter && ...
        ~isPSRowExist(PSRunData, PSLoopingParam.trackingRow)
    % Save tracking row as tracked lead row since following flame lead here...
    PSRunData.circleCenterLeadRow(PSLoopingParam.iCurrentFrame) = ...
        circleCenterLead;
else
    % No change anymore since at the end
    PSRunData.circleCenterLeadRow(PSLoopingParam.iCurrentFrame) = ...
        PSRunData.circleCenterLeadRow(PSLoopingParam.iCurrentFrame-1);
    % PSLoopingParam.trackingRow;
end

%% Save Pin Center and Circle Center flame locations
PSRunData.dumbflameloc_cirCen(PSLoopingParam.iCurrentFrame) = ...
    PSRunData.PSRows(circleCenterLead).dumbflameloc(PSLoopingParam.iCurrentFrame);
PSRunData.dumbflameloc(PSLoopingParam.iCurrentFrame) = ...
    PSRowData.dumbflameloc(PSLoopingParam.iCurrentFrame);

PSRunData.autoflameloc_cirCen(PSLoopingParam.iCurrentFrame) = ...
    PSRunData.PSRows(circleCenterLead).autoflameloc(PSLoopingParam.iCurrentFrame);
PSRunData.autoflameloc_pinCen(PSLoopingParam.iCurrentFrame) = ...
    PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame);

%% Plot Video of Current Flame Locations
% Make looping fig current
set(0,'currentfigure',frameLoopingFigHand)

if PSFlags.dispLoopingPlot || PSFlags.saveFrameStills || PSFlags.saveOutVideo
    % Only plot if seeing or saving the output...
    if PSFlags.saveImgFrameOnly
        subplot(1,1,1)
    elseif ~PSFlags.reduceLoopFeatures || PSFlags.plotBrightnessHistogram
        subplot(2,2,1)
    else % PSFlags.reduceLoopFeatures && ~PSFlags.plotBrightnessHistogram
        subplot(2,1,1)
    end
    plotPSImgWithTicks(frameLoopingFigHand, PSRunData, PSConditions, ...
        PSUsrInput, PSLoopingParam, PSFlags);

    if ~PSFlags.saveImgFrameOnly
        if ~PSFlags.reduceLoopFeatures || PSFlags.plotBrightnessHistogram
```

```
            subplot(2,2,[3,4])
                % Bottom two plot grids
        else
            subplot(2,1,2)
                % Bottom of 2x1 grid
        end
        if ~PSLoopingParam.isFlameStart || ...
                ~isPSRowExist(PSRunData, PSLoopingParam.trackingRow)
            plotPSRowData(frameLoopingFigHand, PSRowData, PSConditions, ...
                PSUsrInput, PSLoopingParam, PSFlags);
        else
            plotPSRowData(frameLoopingFigHand, ...
                PSRunData.PSRows(PSLoopingParam.trackingRow), ...
                PSConditions, PSUsrInput, PSLoopingParam, PSFlags);
        end
    else
        if PSLoopingParam.isFlameStart || ...
                ~isPSRowExist(PSRunData, PSLoopingParam.trackingRow)
            flameDist=(PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)-...
                PSUsrInput.pinsCenter_x)/PSUsrInput.pixel2cm;
            if PSFlags.isPrintFlameCM
                fprintf('\tFrame %d, Flame Traveld: %.2f cm\n', ...
                    PSLoopingParam.iCurrentFrame, flameDist)
            end
        else
            flameDist=...
                (PSRunData.PSRows(PSLoopingParam.trackingRow).autoflameloc(PSLoopingParam.
                     iCurrentFrame)-...
                    PSUsrInput.pinsCenter_x)/PSUsrInput.pixel2cm;
            if PSFlags.isPrintFlameCM
                fprintf('\tFrame %d, Flame Traveld: %.2f cm\n', ...
                    PSLoopingParam.iCurrentFrame, flameDist)
            end
        end
    end

end

%% Save Frames and Video as desired
if PSFlags.saveFrameStills
    saveas(frameLoopingFigHand, ...
        fullfile(strcat(PSConditions.folderPath, filesep, ...
        strcat('PostProcessVideo',filesep,'PPVideo_', PSConditions.VideoName,...
        '_', num2str(PSLoopingParam.iCurrentFrame), '.png'))), 'png');
end
if PSFlags.saveOrigStills
    imwrite(PSLoopingParam.frame, ...
        fullfile(strcat(PSConditions.folderPath, filesep, ...
        strcat('OriginalFrames', filesep, 'Original_', PSConditions.VideoName,...
        '_', num2str(PSLoopingParam.iCurrentFrame),'.png'))));
end
if PSFlags.saveOutVideo
    [PSAllOutputFrames, outputVideoObj, PSLoopingParam] ...
        = writePSOutputLoopingVideo(PSAllOutputFrames, frameLoopingFigHand,...
        outputVideoObj, PSConditions, PSLoopingParam, PSFlags);
end

if PSFlags.dispLoopingPlot && ~PSFlags.useMatlabVideoReader
        pause(.001) % Pause to allow display to catch up
end
```

```
            PSLoopingParam.iCurrentFrame=PSLoopingParam.iCurrentFrame+1;

        end

        %% Cleanup Video Loop Reading
        if PSFlags.saveOutVideo && PSFlags.useMatlabVideoWriter
            if PSFlags.writeVideoAtEndOfLoop
                % If writting at end of loop to save time, now write them!
                % Crop unnecessary frames
                PSAllOutputFrames = PSAllOutputFrames(1:PSLoopingParam.frameVidEnd);
                try
                    writeVideo(outputVideoObj, PSAllOutputFrames)
                catch e
                    fprintf('Errored on video write here\n')
                end
                % Clear so it doesn't save all the frames of the output video
                clear PSAllOutputFrames

            end
            close(outputVideoObj)
        end
        close(frameLoopingFigHand)

        if ~PSFlags.useMatlabVideoReader
            clear PSAllFrames
                % Clear so it doesn't save all the frames of the input video
        end
        % Crop Variables to PSLoopingParam.frameVidEnd instead of PSConditions.numFrames
        PSRunData.dumbflameloc=PSRunData.dumbflameloc(1:PSLoopingParam.frameVidEnd);
        PSRunData.autoflameloc_pinCen=...
            PSRunData.autoflameloc_pinCen(1:PSLoopingParam.frameVidEnd);
        PSRunData.autoflameloc_cirCen=...
            PSRunData.autoflameloc_cirCen(1:PSLoopingParam.frameVidEnd);
        if ~PSFlags.suppressDebugOutput
            fprintf('\tCropped vars to end at %d instead of %d. ', ...
                PSLoopingParam.frameVidEnd, PSConditions.numFrames)
        end

        fprintf('Looping elapsed time: %.1f\n', cputime-PSLoopingStartTime)

    else
        %% Load Previous matlab Session
        fprintf('\n\tLoading previous matlab session instead of looping through frames')
        [PSRunData, PSScopeData, PSConditions, PSUsrInput, PSLoopingParam, PSFlags] = ...
            grabPrevRunData(PSConditions, PSFlags);

    end % End video progression

    if PSFlags.saveFrameStills || PSFlags.saveImgFrameOnly
        fprintf('Done with Saving Video Frames, exiting now...\n')
        return
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %% Write user input from PSConditions and PSUsrInput to file for documenting
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    writePSUsrInput(PSConditions, PSUsrInput, PSFlags);

    fprintf('Wrote key input (within PSConditions, PSUsrInput) to file\n')
    %% Rename output video file with relevant file indicators for easier storing
```

```
if isSortNameVideo && PSFlags.saveOutVideo
   if PSConditions.isBubble || PSConditions.phiInit ~= PSConditions.phiPost
      eqNominalName=sprintf('%.3finto%.3f', phiInit, phiPost);
   elseif PSConditions.phiInit==PSConditions.phiPost
      eqNominalName=sprintf('Homogen_%.3f',phiPost);
   else
      fprintf('Error in determining eqApproxName\n')
   end
   status = system(sprintf('cp %s%sFlame_Propagation_%s.avi %s%sFlame_Prop_%s_%s.avi', ...
      PSConditions.folderPath, filesep, PSConditions.VideoName, ...
      PSConditions.folderPath, filesep, eqNominalName, PSConditions.VideoName), '-echo');
   if status~=0
      fprintf("ERROR in saving process Flame_Prop_* video! Errored in %s\n", ...
         PSConditions.VideoName)
      pause(360)
   else
      fprintf("\tSaved copy of %s with different filename\n", PSConditions.VideoName)
   end
end

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Post Process image data gathered
fprintf('\nInterpreting Gathered Data...\n')
tic
if true
   %% Assemble Gathered Data into Structure
   PSPlotDispProp = initPSPlotDispProperties;
   PSRunData = gatherPSRunData(PSRunData, PSConditions, PSUsrInput, ...
      PSLoopingParam, PSPlotDispProp, PSFlags);

   %% Interpolating/smoothing
   PSPostProcData = interpPSRunData(PSRunData, PSScopeData, PSConditions, ...
      PSUsrInput, PSPlotDispProp, PSFlags);

   PSPlotDispProp.bubbleAxisHeigh=...
      max(PSPostProcData.drdt_5pt_SG(1:...
         int64(PSPlotDispProp.splineResolutionFactor/...
            PSPlotDispProp.splineDisplayReducFactor):end));
   PSPlotDispProp.SGMax=...
      max(PSPostProcData.SR_SG(PSLoopingParam.flameKernalDevelFrameNumb:end))+...
      2*std(PSPostProcData.SR_SG);
      % Compute max stretch rate after flame kernal development for plotting
      % Plotting parameter to plot ~95%
      % first stretch points are very large and throws off plot axis
end
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Plot Pixel changes
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   if PSFlags.dispKeyPlots || PSFlags.savePostPlots
      plotPSPixelChanges(PSRunData, PSConditions, PSFlags);
   end
%% Plot Splined Radius and Velocity vs. Time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if PSFlags.dispKeyPlots || PSFlags.savePostPlots
   plotPSRadiusVsTime(PSPostProcData, PSRunData, PSConditions, PSUsrInput, ...
      PSLoopingParam, PSPlotDispProp, PSFlags);

   plotPSVelcityVsTime(PSPostProcData, PSRunData, PSConditions, PSUsrInput, ...
      PSPlotDispProp, PSFlags);
```

```
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Plot drdt vs. Radius, drdt vs. Stretch rate, stretch rate, etc.
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if PSFlags.dispKeyPlots || PSFlags.savePostPlots
        plotPSVelocityVsRadius(PSPostProcData, PSRunData, PSConditions, ...
            PSUsrInput, PSPlotDispProp, PSFlags)

        plotPSVelocityVsStretchRate(PSPostProcData, PSConditions, PSUsrInput, ...
            PSPlotDispProp, PSFlags);

        plotPSStretchRateVsRadius(PSPostProcData, PSRunData, PSConditions, ...
            PSPlotDispProp, PSFlags);
        plotPSStretchRateVsTime(PSPostProcData, PSRunData, PSConditions, ...
            PSPlotDispProp, PSFlags);
    end



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Plot Combined Scope data: Pressure and matching flame radius data
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if PSFlags.dispKeyPlots || PSFlags.savePostPlots
        plotPSRadiusAndPressureVsTime(PSPostProcData, PSRunData, PSScopeData, ...
            PSConditions, PSUsrInput, PSLoopingParam, PSPlotDispProp, PSFlags)
        plotPSRadiusAndPressure_allVsTime(PSPostProcData, PSRunData, PSScopeData, ...
            PSConditions, PSUsrInput, PSLoopingParam, PSPlotDispProp, PSFlags)
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Write Key outputs to txtfile if looping, changed user input, or saving post plots
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Write key output if changed something...
    if PSFlags.isRunLoop || ~PSFlags.debug_skipUsrInput || PSFlags.savePostPlots
        writePSKeyOutput(PSPostProcData, PSRunData, PSScopeData, PSConditions, ...
            PSUsrInput, PSLoopingParam, PSPlotDispProp, PSFlags)
        fprintf('WROTE KEY OUTPUTS TO FILES\n')
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Clean Up and Save Vars, always saving
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    clear usrInputFrameFigHand figHandPixDiff figHandSpline figHandRvT ...
        figHandDrdtVT figHandSrvDrdt figHandSrvDrdt frameLoopingFigHand ...
        schfig figHandDrdtVR singleFrameFigHand figHandRnPvT h
       %reduce size of matlab variables, removing figure handles and other large data info

    if ~isfield(PSConditions, 'specificRunMatlabFN')
        PSConditions.specificRunMatlabFN = fullfile(strcat(PSConditions.folderPath, ...
            filesep,'MatlabSessionVars_', PSConditions.VideoName,'.mat'));
    end
    save(PSConditions.specificRunMatlabFN)
    %% Completed Note
    fprintf('Completed processing %s in %.0f sec.\n', PSConditions.VideoName, ...
        cputime-funcStartTime) % Pauseing for long run at end of process script file

catch e % Catch the MException struct for printing
    %% Print error message
    fprintf('Caught Error at end of Bubble looping: %s Occured in post-Processing %s\n', ...
        e.identifier, PSConditions.VideoName)
    fprintf('\t%s\n', e.message)

    fprintf('\n\nProcessSchlieren_Bubble.m ERROR: %s\n\n', PSConditions.VideoName)
    if strcmp(e.identifier, 'MATLAB:handle_graphics:exceptions:UserBreak') ... % ctr-c
            || strcmp(e.identifier, 'MATLAB:structRefFromNonStruct') % close looping window
        % If user is trying to kill job
        fprintf('STOPPING: Recieved user input to stop.\n')
        return; %rethrow(e)
    else % Print everything without rethrowing error...
        getReport(e,'extended','hyperlinks','on')
    end
    %% Do something about the error...
    %rethrow(e)
    if ~strcmp(e.identifier, 'MATLAB:handle_graphics:exceptions:UserBreak') ... % ctr-c
        && ~strcmp(e.identifier, 'MATLAB:structRefFromNonStruct') % closes looping window
        % If not user input error...
        writePSRunError(PSConditions, e)
    end
end

pause(0) % 0 for 2, 5 for 3, maybe 10 for 4...
```

Get Row Relevant Locations: getPSRowFlameLoc

```
function [PSRowData] = getPSRowFlameLoc(j_frameCol, PSRowData, PSRunData, ...
    PSConditions, PSUsrInput, PSLoopingParam, PSFlags)
% Detect the flame location, ignition
%
% Find flame location using dumb and auto ways given input conditions, frame shown as:
% frame(col, row) Black is 0, 255 is white

% Check PSRowData j_frameCol
if PSRowData.frameCol ~= j_frameCol
    fprintf('Error in j Frame and PSRowData\n')
    pause(10)
end

% Determine plausible section to look in to reduce spline load
%% Grab Row Data from PSLoopingParam.frame
PSRowData = getPSLoopingParamUpdateRow(PSRowData, PSRunData, PSConditions, ...
    PSUsrInput, PSLoopingParam);

%% Dumb Flame Tracker Flame Loc
PSRowData = getPSRowDumbFlameLoc(PSRowData, PSConditions, PSUsrInput, PSLoopingParam);

%% Bubble Tracker Loc
PSRowData = getPSRowBubbleLoc(PSRowData, PSConditions, PSUsrInput, PSLoopingParam);

%% Auto Flame Tracker Flame Loc
PSRowData = getPSRowAutoFlameLoc(PSRowData, PSRunData, PSConditions, PSUsrInput, ...
    PSLoopingParam, PSFlags);
```

Get Row Flame Location: getPSRowAutoFlameLoc

```
function PSRowData = getPSRowAutoFlameLoc(PSRowData, PSRunData, PSConditions, PSUsrInput, ...
    PSLoopingParam, PSFlags)

%% Determine if in bubble popping frame or already at the edge of window
```

```
NumbFramesHold=1;
if (PSRowData.BubblePopedFrame && (PSRowData.BubblePopedFrame > ...
       PSLoopingParam.iCurrentFrame - NumbFramesHold) ) ...
       && (PSRowData.autoBubbleloc(PSRowData.BubblePopedFrame-1)-...
       PSRowData.autoflameloc(PSRowData.BubblePopedFrame-1)...
           <=PSLoopingParam.unPhysicalPixJump)
    % If within X frames of popping bubble close to flame that it may distort tracker...
    % Temporarily inhibit jump
    PSRowData = getPSRowBubbleHoldAdvancement(PSRowData, PSRunData, PSConditions, ...
        PSUsrInput, PSLoopingParam, PSFlags, 0.6);
    return
elseif PSRowData.autoHitEdge
    % If already hit edge, speed up run and skip point...
    PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)=...
        PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
    return
end

%% Try to find flame front from 1D auto method.
useFindPeaks=false;
if ~useFindPeaks
    % Instead, setting to 1 since looking at smoothed with cut-off
    maxIdx = 1;
    maxThreshedIdx_Full = 1;
    autoPickedPixelslocFull_uncut=1;
    autoPickedPixelslocFull_validIdxs=1;
    PSRowData.autoPickedPixelslocFull=1;
    maxThreshedIdx=1;
else
    % Determine possible flame locations
    [~,maxIdx] = findpeaks(-PSRowData.rowValOfInterest_Smoothed);
        % Finds local minima and their index by finding the max of the inverse...
    maxThreshedIdx_Full = maxIdx(PSRowData.rowValOfInterest_Smoothed(maxIdx)...
        < PSConditions.autoFlameLocMinBrtThresh);
        % Find minima's index of index maxIdx, to Find minima lower than a threshold
    maxIdx = [1; maxIdx]; % always check the first point as search point
    maxThreshedIdx_Full = [1; maxThreshedIdx_Full];

    autoPickedPixelslocFull_uncut=PSRowData.rowValOfInterest_Smoothed(maxThreshedIdx_Full);
    if PSLoopingParam.iCurrentFrame>PSRowData.flameKernalDevelFrameNumb
        % If outside the sparking window, ie no longer in flame kernal development
        autoPickedPixelslocFull_validIdxs=autoPickedPixelslocFull_uncut <= ...
            PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
    else
        autoPickedPixelslocFull_validIdxs=autoPickedPixelslocFull_uncut <= ...
            PSUsrInput.pixEdgeOfRIO;
    end
    PSRowData.autoPickedPixelslocFull=...
        autoPickedPixelslocFull_uncut(autoPickedPixelslocFull_validIdxs);
    maxThreshedIdx = maxThreshedIdx_Full(autoPickedPixelslocFull_validIdxs);
end

%% Check if starting or erroring (whether to keep at pin center location)
% Pick last pixel and follow to find point of a specified darkness, do not grab the bubble
if isempty(PSRowData.autoPickedPixelslocFull)
    % If in beginning or in an error, autoHitEdge captures the latter
    autoPickedPixelsloc=PSUsrInput.pinsCenter_x;
    PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)=autoPickedPixelsloc;

    if ~PSRowData.isFlameStart
```

```
        PSRowData.testVectEndIdx=1;
    else
        if PSUsrInput.pixEdgeOfRIO - PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1) ...
                < 2*PSLoopingParam.unPhysicalPixJump
            % If near PSUsrInput.pixEdgeOfRIO
            fprintf(['Errored in finding autoPickedPixelslocFull, empty.',...
                '\nSet to pinsCenter_x, but reduce PSUsrInput.pixEdgeOfRIO in next run\n'])
        else
            % Else flame tracking lost....or possibly a mis-fire and can ignore.
        end
    end
    return
end
if ~PSUsrInput.isRightDark
    %look to the left for drop to darknessSelection
    if PSUsrInput.pixEdgeOfRIO < PSUsrInput.pinsCenter_x
        fprintf('LEFT SIDE NOT FLIPPED, need to reflect/rotate and continue...\n')
        pause(10)
    end
end

%% Check each possible flame location
% While loop to find index for NEarest Detected maxThreshed.
% Subtract until reach point where jump is real
idxForNearestDetectedIdx=length(maxThreshedIdx);
    % Initial guess is try with end of local mins
iWhile=1;
while true && iWhile<=length(maxThreshedIdx)
    % Do while loop hack...
    isSubLocPastPhysical=false;
        % Set at each iteration for finding the correct local min to use
    PSRowData.nearestDetectedIdx=maxThreshedIdx(idxForNearestDetectedIdx);
        % Try another

    %% Auto Adjust brightnessThickness loop
    % Keep advancing along rowValOfInterest curve until hit the first darknessSelection
    if PSRowData.isFlameStart
        bubbleExpansionTol=50;
        pixBubbleLocTolBefore=0;
        pixBubbleLocTolAfter=bubbleExpansionTol;

        rowHadBub= abs(int64(PSUsrInput.pinsCenter_y)-PSRowData.frameCol) <= ...
            int64(PSUsrInput.pinsCenter_y)+bubbleExpansionTol;
        if PSConditions.isBubble
            initBubLoc_row=sqrt(...
                (PSUsrInput.pixel_bubble_right_auto-PSUsrInput.bubbleCenter_x_pixLoc)^2 ...
                -double(int64(PSUsrInput.bubbleCenter_y_pixLoc)-PSRowData.frameCol)^2 ) +...
                PSUsrInput.bubbleCenter_x_pixLoc; % Using true bubble center
        else
            % Assuming circle centered at pins center with radius pixel_bubble_right_avg
            initBubLoc_row=sqrt(...
                (PSUsrInput.pixel_bubble_right_avg-PSUsrInput.pinsCenter_x)^2 ...
                -double(int64(PSUsrInput.pinsCenter_y)-PSRowData.frameCol)^2 ) + ...
                PSUsrInput.pinsCenter_x;
        end
        if PSConditions.isBubble && ~PSRowData.BubblePopedFrame && rowHadBub....
                && ( (PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1)>=...
                    initBubLoc_row-pixBubbleLocTolBefore) ...
                    && (PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1)<...
                    initBubLoc_row+pixBubbleLocTolBefore) )
            % Flame within initial bubble region and bubble hasn't poped yet
            % need additional assistance as flame merges with bubble
            brightBonusStart= 4;%abs(PSRowData.initialMean-PSRunData.leadPSRow.initialMean);
        else
            brightBonusStart=0;
        end
        initBrightnessThickness=1 + brightBonusStart;
        % thickness to just catch 1 pixel, +/- adjusts automatically in while loop below,
        % with an additional boost depending on brightness differences with the lead
    else
        % Increased from 1 to capture steep flame fronts...
        initBrightnessThickness=5;
    end
    brightnessThickness=initBrightnessThickness;
    logicBrightnessTestVect=[0 0]; % legnth=2 to get the while loop running
    whIterNum=0;
    if PSLoopingParam.iCurrentFrame<=PSRowData.flameKernalDevelFrameNumb
      % If trying to catch the initial flame, lower brightness to reduce noFlame workload
      if ~PSRowData.isLead && ~PSRowData.isFlameStart && PSRunData.leadPSRow.isFlameStart
          % Test whether it is early or late lagging flame start
          if (PSRunData.leadPSRow.flameKernalDevelFrameNumb > ...
                  PSRowData.flameKernalDevelFrameNumb - 10)
              % If starting around the time as the lead flame, no modifications necessary
              brightAdjIterMax=100; % 100 allowed for early start
          else
              % Else if trying to start tracking a long time after
              brightAdjIterMax=30;
          end
          % If lead flame started, but lagging rows haven't started,
          % increase brightness adjust to catch known existing flame
      else
          brightAdjIterMax=10; % limit to keep it tight at the start of flame kernel
      end
    elseif PSRowData.autoHitEdge
        brightAdjIterMax=255; % no limit when at the end...
    else
        brightAdjIterMax=100; % Expect to be close along flame path
    end
    while ~PSRowData.autoHitEdge && nnz(logicBrightnessTestVect)<1 ...
            && whIterNum <= brightAdjIterMax
        % count number of non-zeros to see if incresaing brightnessThickness is a good value
        % Keep small as possible to not grab the wrong point that is darker than the value
        if PSRowData.isFlameStart
            % Looking instad of until end of array, look only until unPhysicalPixJump....
            % Find unphysical pixel location, using rowValOfInterest_Smoothed
            if PSLoopingParam.iCurrentFrame>PSRowData.flameKernalDevelFrameNumb
                %allow some slip with initial 20 or so frames for flame kernal development
                idxUnphysicalS = find( PSRowData.ipixLocOfInterest_Smoothed(...
                        PSRowData.nearestDetectedIdx:end) ...
                   >= PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1) +...
                   PSLoopingParam.unPhysicalPixJump);
            else
                idxUnphysicalS = find( or(...
                    PSRowData.ipixLocOfInterest_Smoothed(PSRowData.nearestDetectedIdx:end) ...
                       >= PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1) + ...
                       PSRowData.initUnPhysPixJump,...
                    PSRowData.ipixLocOfInterest_Smoothed(PSRowData.nearestDetectedIdx:end) ...
                        > PSRowData.iPixEndOfRow-2*PSLoopingParam.unPhysicalPixJump) );
                    % Additional flame jump not lead flame
                    % but limit to not jump to edge of window in this vulnerable point
            end
            if isempty(idxUnphysicalS)
                % Changing for reduced smoothed, since if no unphysical points
                % (cut them off before smoothing) it gives an unphysical number here
                PSRowData.testVectEndIdx=length(PSRowData.ipixLocOfInterest_Smoothed);
            else
                PSRowData.testVectEndIdx=idxUnphysicalS(1)-1+PSRowData.nearestDetectedIdx;
                    % use instead of "end"
                if idxUnphysicalS(1) == 1
                    isSubLocPastPhysical=true;
                end
            end
        else %if i small, before flame is detected...
            % Find unphysical pixel location
            PSRowData.nearestDetectedIdx=1;
            if PSRowData.possiblySparked
                idxUnphysicalS = find( or(...
                    PSRowData.ipixLocOfInterest_Smoothed(PSRowData.nearestDetectedIdx:end) ...
                        >= PSRowData.possiblySparked + PSRowData.initUnPhysPixJump,...
                    PSRowData.ipixLocOfInterest_Smoothed(PSRowData.nearestDetectedIdx:end) ...
                        > PSRowData.iPixEndOfRow-2*PSLoopingParam.unPhysicalPixJump));
            else
                idxUnphysicalS = find( or(...
                    PSRowData.ipixLocOfInterest_Smoothed(PSRowData.nearestDetectedIdx:end) ...
                        >= PSUsrInput.pinsCenter_x + PSRowData.initUnPhysPixJump,...
                    PSRowData.ipixLocOfInterest_Smoothed(PSRowData.nearestDetectedIdx:end) ...
                        > PSRowData.iPixEndOfRow-2*PSLoopingParam.unPhysicalPixJump));
            end

            if isempty(idxUnphysicalS)
               % If close to the end, just use the end as end
               idxUnphysicalS = length(PSRowData.ipixLocOfInterest_Smoothed);
            end
            PSRowData.testVectEndIdx=idxUnphysicalS(1)-1+PSRowData.nearestDetectedIdx;
                % use instead of "end"

        end
        idxRangeOfrowVals=PSRowData.testVectEndIdx-PSRowData.nearestDetectedIdx;

        %%%%%% Need to look only in physically acceptable region
        logicBrightnessTestVect=and(...
            PSRowData.rowValOfInterest_Smoothed(PSRowData.nearestDetectedIdx:...
                    PSRowData.testVectEndIdx) > ...
                (PSRowData.flameCutoffThreshold-brightnessThickness), ...
            PSRowData.rowValOfInterest_Smoothed(PSRowData.nearestDetectedIdx:...
                    PSRowData.testVectEndIdx) < ...
                (PSRowData.flameCutoffThreshold+brightnessThickness)...
            ); % Adding flameSlipTolerance for looking at current step to prevent jumps
        %%%%%% Need to look only in physically acceptable region
        if PSLoopingParam.iCurrentFrame==1
            break; % Will fail here as flame doesn't exist here!
        elseif isSubLocPastPhysical
            % Sub point is too far forward, no logical points exist! Try a different point!
            break;
        elseif whIterNum == brightAdjIterMax-1
            if sum(logicBrightnessTestVect) ==0
                brightnessThickness=brightnessThickness+1;
                % Increase as want to have at least 1 valid point!
            elseif sum(logicBrightnessTestVect) ==2
                % Do nothing if oscilating between 2 values and already at the 2 point mark
                break; % Break as want these 2 points
            end
        elseif sum(logicBrightnessTestVect) == 0
            %if brightnessThickness too small
            brightnessThickness=brightnessThickness+1;
        elseif sum(logicBrightnessTestVect) > 1 && brightnessThickness > initBrightnessThickness
            % brightnessThickness too large, but make sure not last iter,
            % don't end here if not a point that exactly gives 1 point
            brightnessThickness=brightnessThickness-1;
        else
            %Do nothing as found it! Will exit while loop on own
        end
        whIterNum=whIterNum+1; %escape method in runaway cases!
    end %% Auto Adjust brightnessThickness loop

    %% Test Brightness Loop
    if whIterNum >= brightAdjIterMax && sum(logicBrightnessTestVect)==2
        % Ok, as have two choices to choose between
    elseif whIterNum >= brightAdjIterMax && sum(logicBrightnessTestVect)>2
        %If ran on and off...and more than 2 (2 case well debugged)
    elseif sum(logicBrightnessTestVect)~=1 && whIterNum >= brightAdjIterMax ...
            && idxForNearestDetectedIdx==1
        if PSRowData.isFlameStart
            % else Spark likely not initiated yet...
        end
    end

    %% Advance with flameIdxAutoS
    flameIdxAutoS = find(logicBrightnessTestVect);
    if PSRowData.isFlameStart && ~PSRowData.autoHitEdge && PSLoopingParam.iCurrentFrame > ...
            PSRowData.flameKernalDevelFrameNumb &&...
            ~isempty(flameIdxAutoS) && length(flameIdxAutoS)>1 ...
            && ((PSConditions.isBubble ... % ~PSConditions.isBubble ||
                && (PSRowData.BubblePopedFrame==0 || PSRowData.BubblePopedFrame && ...
                (PSLoopingParam.iCurrentFrame<=PSRowData.BubblePopedFrame+5)))) % And no bubble
        % If in flame regime and no longer in flame devel region,
        % Assuming there is still a bubble to watch out for
        flameIdxAuto=getPSClosestValidIdx(PSRowData, PSLoopingParam, flameIdxAutoS, PSFlags);
    elseif PSRowData.isFlameStart && ~PSRowData.autoHitEdge && PSLoopingParam.iCurrentFrame...
            > PSRowData.flameKernalDevelFrameNumb &&...
            ~isempty(flameIdxAutoS) && length(flameIdxAutoS)>1 ...
            && ~PSConditions.isBubble || (PSConditions.isBubble ...
            && PSRowData.BubblePopedFrame ...
            && PSLoopingParam.iCurrentFrame>PSRowData.BubblePopedFrame+5) % Still a bubble
        % If in flame regime and no longer in flame devel region, And no bubble issues
        flameIdxAuto=getPSFarthestValidIdx(PSRowData, PSLoopingParam, flameIdxAutoS, PSFlags);
    elseif PSRowData.isFlameStart && ~PSRowData.autoHitEdge ...
            && PSLoopingParam.iCurrentFrame <= PSRowData.flameKernalDevelFrameNumb &&...
            ~isempty(flameIdxAutoS) && length(flameIdxAutoS)>1
        % If flame developing choose farthest rising edge, so start from end and check backwards
        flameIdxAuto=getPSFarthestValidIdx(PSRowData, PSLoopingParam, flameIdxAutoS, PSFlags);
    elseif ~PSRowData.isFlameStart && length(flameIdxAutoS)>1
        if nnz(flameIdxAutoS) >=2 && ( ...
                (PSConditions.isBubble && ~PSRowData.BubblePopedFrame ... % Bubble and exists
                        && ( PSRowData.ipixLocOfInterest_Smoothed(flameIdxAutoS(end)) ...
                                < PSUsrInput.pixel_bubble_right_auto ...
                            && (PSRowData.ipixLocOfInterest_Smoothed(flameIdxAutoS(end)) ...
                                < getPSMinNearBubPt(PSRowData, PSLoopingParam)...
                                    -PSLoopingParam.unPhysicalPixJump) ) )...
                || ( (~PSConditions.isBubble || PSConditions.isBubble ...
                && PSRowData.BubblePopedFrame) ... % If there isn't a bubble
                        && (idxUnphysicalS(1) ~= length(PSRowData.ipixLocOfInterest_Smoothed)) ))
                % If only 1 element, then check...will be too high to ever be a problem
            flameIdxAuto=getPSFarthestValidIdx(PSRowData, PSLoopingParam, flameIdxAutoS, PSFlags);

        elseif nnz(flameIdxAutoS) >=2 && (~PSConditions.isBubble || PSConditions.isBubble ...
                && PSRowData.BubblePopedFrame) && length(idxUnphysicalS)==1 ... % No bubble
                && idxUnphysicalS == length(PSRowData.ipixLocOfInterest_Smoothed )
            % If ~isFlameStart && length(flameIdxAutoS)>1 and If greather than 2 points found,
            % that aren't cut for unphysical since expanded...
            % so have included the end and need to crop off unphysical points
            if max(PSRowData.autoflameloc)==PSUsrInput.pinsCenter_x
                % If never have had a flame propigating aka only at pin center...
                priorValidSolidCutoffPt=...
                    PSRunData.leadPSRow.autoflameloc(PSLoopingParam.iCurrentFrame) ...
                    + PSLoopingParam.unPhysicalPixJump; % Then use lead flame
            else
                % Else use prior flame
                priorValidSolidCutoffPt=max(PSRowData.autoflameloc) ...
                    + 2*PSLoopingParam.unPhysicalPixJump;
            end
            subIdxFlameIdxAutoS = and(...
                PSRowData.ipixLocOfInterest_Smoothed(flameIdxAutoS) ...
                    < (PSRunData.leadPSRow.autoflameloc(PSLoopingParam.iCurrentFrame)...
                        +PSLoopingParam.unPhysicalPixJump), ...
                ~and(... % And cut off end solid body thresholded values
                    PSRowData.ipixLocOfInterest_Smoothed(flameIdxAutoS) ...
                        > priorValidSolidCutoffPt,... % solid body after fatherest flame loc
                    PSRowData.rowValOfInterest_Smoothed(flameIdxAutoS) ...
                        < PSConditions.SolidBodyThreshold ));
                % Had to remove threshold of solid body threshold
            if nnz(subIdxFlameIdxAutoS)>0
                tmpFlameIdxAutoS = flameIdxAutoS(subIdxFlameIdxAutoS);
                % Use end of possible points, restricted by the lead flame tracked point
                flameIdxAuto=getPSFarthestValidIdx(PSRowData, PSLoopingParam, ...
                    tmpFlameIdxAutoS, PSFlags);
            else
                % Else no valid points, skipping...
                flameIdxAuto=[];
            end
        elseif PSConditions.isBubble && ~PSRowData.BubblePopedFrame ... % Bubble exists
                        && ( PSRowData.ipixLocOfInterest_Smoothed(flameIdxAutoS(end)) ...
                            >= PSUsrInput.pixel_bubble_right_auto ...
                            || ( PSRowData.ipixLocOfInterest_Smoothed(flameIdxAutoS(end)) ...
                                >= getPSMinNearBubPt(PSRowData, PSLoopingParam)...
                                    -PSLoopingParam.unPhysicalPixJump) )
            % ~isFlameStart && length(flameIdxAutoS)>1 and if have included points near bubble,
            % need to crop off bubble points if not in the possible flame region
            subIdxFlameIdxAutoS = or( ...
                PSRowData.ipixLocOfInterest_Smoothed(flameIdxAutoS) ...
                        < getPSMinNearBubPt(PSRowData, PSLoopingParam)...
                        -PSLoopingParam.unPhysicalPixJump, ...
                PSRowData.ipixLocOfInterest_Smoothed(flameIdxAutoS) ...
                        <= max(PSRowData.autoflameloc) + PSLoopingParam.unPhysicalPixJump);
                % To allow for approach of bubble when nearby
                % Crop off bubble points, and those near it!
            if nnz(subIdxFlameIdxAutoS)>0
                tmpFlameIdxAutoS = flameIdxAutoS(subIdxFlameIdxAutoS);
                flameIdxAuto=getPSFarthestValidIdx(PSRowData, PSLoopingParam, ...
                        tmpFlameIdxAutoS, PSFlags);
            else
                % Else no valid points, skipping...
                flameIdxAuto=[];
            end
        else
            flameIdxAuto=flameIdxAutoS(1);
        end
        % If detected more than 1 point before flame detected, just select first point...
    else
        flameIdxAuto=flameIdxAutoS;
    end

    %% Test flameIdxAuto
    if ~PSRowData.isFlameStart && isempty(flameIdxAuto)
        % If failed because flame hasn't started...
        flameIdxAuto=1;
    elseif PSRowData.autoHitEdge
        %hit edge already...
        autoPickedPixelsloc=PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
        break;
    elseif isempty(flameIdxAuto)
        % If failed to detect a physical flame...
    end

    % Check for special case that need not continue,first or hit the edge
    if PSLoopingParam.iCurrentFrame==1
        % If first case, PSRowData.autoflameloc doesn't have any values!
        autoPickedPixelsloc=PSUsrInput.pinsCenter_x; % Set as will be empty
        pixJump=autoPickedPixelsloc-PSUsrInput.pinsCenter_x;
        break; % Break out of while loop as i=1 case is special...
    elseif PSRowData.autoHitEdge
        break; % Flame tracking has ended
    end

    % Check if detection failed previously
    if isempty(flameIdxAuto) || isSubLocPastPhysical
         % Other logical tests fail below...for debugging
        % Local min in location not flame detectable
        idxForNearestDetectedIdx=idxForNearestDetectedIdx-1;
            % Go into while loop with a previous local min - jumped passed a bubble
        if idxForNearestDetectedIdx<1
            idxForNearestDetectedIdx=1;
        end
    else
        % Else no prior errors detected, so continue
        % Determine the automatic detection pixel location,
        % autoPickedPixelsloc=ipixLocOfInterest(flameIdxAuto+nearestDetectedIdx-1);
        autoPickedPixelsloc=...
            PSRowData.ipixLocOfInterest_Smoothed(flameIdxAuto+PSRowData.nearestDetectedIdx-1);

        % Determine pixJump to ensure physical!
        pixJump=autoPickedPixelsloc-PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
        if isempty(pixJump)
            % Skip this point as it causes other logical test to fail...but unknown why
            fprintf('\t i=%d Error: pixJump is empty but flameIdxAuto isnt\n', ...
                PSLoopingParam.iCurrentFrame)
        end

        % Brightness rising slope test
        if PSRowData.isFlameStart
            brtVal_FlameSelected=PSRowData.rowValOfInterest_Smoothed(...
                flameIdxAuto+PSRowData.nearestDetectedIdx-1);
            brtVal_LeftOfFlameSelected=PSRowData.rowValOfInterest_Smoothed(...
                max(flameIdxAuto+PSRowData.nearestDetectedIdx-1 -1, 1));
            brtVal_RightOfFlameSelected=PSRowData.rowValOfInterest_Smoothed(...
                min(flameIdxAuto+PSRowData.nearestDetectedIdx-1 +1, ...
                length(PSRowData.rowValOfInterest_Smoothed)) );
        end
        % Process jump in pixel
        if (PSLoopingParam.iCurrentFrame>PSRowData.flameKernalDevelFrameNumb && pixJump ...
                    > PSLoopingParam.unPhysicalPixJump) || ...
                PSLoopingParam.iCurrentFrame<=PSRowData.flameKernalDevelFrameNumb && ...
                    ( (PSRowData.possiblySparked && (pixJump > PSRowData.initUnPhysPixJump ...
                        + (PSRowData.possiblySparked-PSUsrInput.pinsCenter_x))) ...
                        || (~PSRowData.possiblySparked && pixJump ...
                            > PSRowData.initUnPhysPixJump) )
            % if not physical or pixel jump execeds physical limitations
            if PSRowData.isFlameStart
                % Local min in location not flame detectable
                idxForNearestDetectedIdx=idxForNearestDetectedIdx-1;
                    % Go into while loop with a previous local min likly jumped passed bubble
                if idxForNearestDetectedIdx<1
                    idxForNearestDetectedIdx=1;
                end
            else
                % Skip this point, no flame yet detected
            end
        elseif ~PSRowData.isFlameStart
            if pixJump > 0 && ( (PSRowData.possiblySparked ...
                    && (pixJump <= PSRowData.initUnPhysPixJump ...
                        + (PSRowData.possiblySparked-PSUsrInput.pinsCenter_x))) ...
                        || (~PSRowData.possiblySparked && pixJump ...
                        <= PSRowData.initUnPhysPixJump) )
                % If flame start has just been detected!
                if ~PSRunData.leadPSRow.isFlameStart ...
                        && PSRowData.possiblySparked ...
                        && autoPickedPixelsloc<=PSRowData.possiblySparked ...
                        || PSRunData.leadPSRow.isFlameStart && (autoPickedPixelsloc ...
                            < max(.5*(PSRunData.leadPSRow.autoflameloc(...
                                PSLoopingParam.iCurrentFrame)-PSUsrInput.pinsCenter_x), 10)...
                                +PSUsrInput.pinsCenter_x )
                    PSRowData.possiblySparked=0;
                    break; % Break as have not detected spark yet
                elseif ~PSRowData.possiblySparked
                    PSRowData.possiblySparked=autoPickedPixelsloc;
                    % Flip flag so next time here it sparked!
                    break; % Break out of loop as flame possibly initialized
                else
                    PSRowData.isFlameStart=true;
                    PSRowData.flameKernalDevelFrameNumb=PSLoopingParam.iCurrentFrame...
                        +PSLoopingParam.flameKernalDevelFrameAdjust/3600*PSConditions.fps;
                    % frame number to start limiting flame speed (allows flame development)
                    break; %Break out of loop as flame just initialized
                end
            else
                % Skip this point as flame not yet detected
                % Local min in location not flame detectable
                idxForNearestDetectedIdx=idxForNearestDetectedIdx-1;
                    % Go into while loop with a previous local min as must have jumped bubble
                if idxForNearestDetectedIdx<1
```

```
                idxForNearestDetectedIdx=1; % For debugging
            end
        end
    elseif pixJump < 0
        % Detected Flame point retreating
        break;
    elseif pixJump == 0
        break; % Flame hasn't moved!
    elseif pixJump > 0
        break;
        % Normal pixJump?
    end
end
iWhile=iWhile+1;
end

% Check flame is advancing, otherwise hit edge
try
    if ~exist('autoPickedPixelsloc', 'var')
        autoPickedPixelsloc=[]; % set to empty
    end
    if isempty(autoPickedPixelsloc) && ( length(...
            PSRowData.startIdxPt(PSRowData.startIdxPt>1))) < 10 && ~PSRowData.isLead
        % If doesn't exist and hasn't existed, then flame is not started and needs to be reset...
        PSRowData.misFire = true; % Flame reset flag

    elseif ~isempty(autoPickedPixelsloc) ...
            && nnz(PSRowData.autoflameloc-PSUsrInput.pinsCenter_x)>1 ...
            && PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1)>autoPickedPixelsloc...
                +PSLoopingParam.flameSlipTolerance
    end
catch e % Catch the MException struct for printing
    fprintf('Error in checking flame advancement: %s Occured in post-Processing %s\n',...
        e.identifier, PSConditions.VideoName)
    fprintf('\t%s\n', e.message)
    rethrow(e)
end

%% Test whether or not to save point
try
    if ~PSRowData.isLead && ~PSRowData.misFire && PSRowData.isFlameStart ...
            && ~PSRowData.autoHitEdge && ( isempty(autoPickedPixelsloc) ...
                || ( pixJump>PSLoopingParam.stallThresh ...
                    && (abs(autoPickedPixelsloc-PSLoopingParam.stallThresh) ...
                        <= PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1)) ) ...
                || ( (PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1)...
                        ~=PSUsrInput.pinsCenter_x) ...
                    && PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1) ...
                    == (PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-2))))
        % Or last 2 frames are identical...possibly stalled on window blemish...
        % Or past two points were held const..but not held at pin center (past failure)
        PSRowData = getPSRowLostFrameAction(PSRowData, PSLoopingParam);

    end
catch e
    fprintf('Errored on testing if lost flame\n')
end
if PSLoopingParam.iCurrentFrame==1
        PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)=PSUsrInput.pinsCenter_x;
```

```
elseif PSRowData.autoHitEdge || ~PSRowData.misFire && ~PSRowData.isFlameStart
    % If already hit edge, errors can be ignored and keep stationary
    PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)=...
        PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);

elseif PSRowData.misFire
    PSRowData.isFlameStart=false;
    PSRowData.possiblySparked = 0;
        % Not flame start, previous was miss-categorized
    % Correct current and previous time step
    PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1:...
        PSLoopingParam.iCurrentFrame)=PSUsrInput.pinsCenter_x;
    PSRowData.flameKernalDevelFrameNumb = PSLoopingParam.initflameKernalDevelFrameNumb;
    PSRowData.misFire = false;
        % Reset missfire for next run
elseif isempty(autoPickedPixelsloc)
    PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)=...
        PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
elseif autoPickedPixelsloc < PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1) ...
        - PSLoopingParam.flameSlipTolerance ||...%slipped backward
        PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1)...
            >= PSUsrInput.pixEdgeOfRIO ||...% went to end
        PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1)...
            >= PSRowData.iPixEndOfRow -2*PSLoopingParam.unPhysicalPixJump ||... % near edge
        PSLoopingParam.iCurrentFrame>PSRowData.flameKernalDevelFrameNumb ...
        && autoPickedPixelsloc > PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1) ...
            + PSLoopingParam.unPhysicalPixJump ||...% Unphysical jump forward
        PSLoopingParam.iCurrentFrame<=PSRowData.flameKernalDevelFrameNumb && ...
            ( (PSRowData.possiblySparked ...
                && (pixJump > PSRowData.initUnPhysPixJump ...
                    + (PSRowData.possiblySparked-PSUsrInput.pinsCenter_x))) ...
                        || (~PSRowData.possiblySparked ...
                            && pixJump > PSRowData.initUnPhysPixJump) ) % Unphysical jump
    % See if hit edge
    pixLocOfMinFlameTravelDistance=PSUsrInput.pixEdgeOfRIO;%400;
    if PSRowData.isFlameStart && (PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1) ...
            >= pixLocOfMinFlameTravelDistance) || PSRowData.isFlameStart ...
                && (PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1) ...
                    >= PSRowData.iPixEndOfRow-2*PSLoopingParam.unPhysicalPixJump)
        % If in a region likely to have reached the end of the flame propigation
        if ~PSRowData.autoHitEdge
            PSRowData.autoHitEdge=true;
            PSRowData.frameVidEnd=min(PSLoopingParam.iCurrentFrame...
                +PSLoopingParam.frameVidEnd_flameEndAdd, PSConditions.numFrames);
        end
    end
    if PSRowData.autoHitEdge
        % If hit edge
        PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)=...
            PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
    elseif autoPickedPixelsloc > PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1) ...
            + PSLoopingParam.unPhysicalPixJump
        % Auto detected flame advanced more than physically possible...
        if PSLoopingParam.iCurrentFrame>PSRowData.flameKernalDevelFrameNumb
            PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)=...
                PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
        else
            PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame) = autoPickedPixelsloc;
        end
    elseif PSConditions.isBubble && autoPickedPixelsloc ...
```

```
            < PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1) ...
                - PSLoopingParam.flameSlipTolerance
        PSRowData = getPSRowBubbleHoldAdvancement(PSRowData, PSRunData, PSConditions, ...
            PSUsrInput, PSLoopingParam, PSFlags, 0.75);

        if ~PSRowData.isLead && ~PSRowData.misFire && PSRowData.isFlameStart ...
                && ~PSRowData.autoHitEdge
            % If not going to screw up the rest of the flame tracking
            PSRowData = getPSRowLostFrameAction(PSRowData, PSLoopingParam);
            % Increment lost frame advancement to prepare for reset if needed
        end
    else
        % All other unphysical things should be saved by leaving flame statioary
        PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)=...
            PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
    end

else %everything ok to save point
    PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame) = autoPickedPixelsloc;
    idxAutoLocBrightness = dsearchn(PSRowData.ipixLocOfInterest_Smoothed, autoPickedPixelsloc);
    PSRowData.autoFlameLocBrightNess(PSLoopingParam.iCurrentFrame) = ...
        PSRowData.rowValOfInterest_Smoothed(idxAutoLocBrightness);
end
```

Get Row Bubble Location: getPSRowBubbleLoc

```
function PSRowData=getPSRowBubbleLoc(PSRowData,PSConditions,PSUsrInput,PSLoopingParam)
% Finds location of bubble, if exists and finds when it pops
% pins center when can't find bubble
% Looks at PSConditions.bubbleCutoffThresh for initial darkness at pin sparking
% extrapolates from it to track the bubble
if ~PSConditions.isBubble || PSRowData.BubblePopedFrame
    % If no bubble to detect, or if already detected a popped bubble
    PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame)=PSUsrInput.pinsCenter_x;
    return % Set default and return
end
PSRowData.bubbleStartLoc=PSRowData.autoflameloc(max(PSLoopingParam.iCurrentFrame-1,1))+...
    PSLoopingParam.unPhysicalPixJump;
pixBubSearchRange=int64(PSRowData.bubbleStartLoc):int64(PSRowData.bubbleEndSearchLoc);
    % Weigh right avg much more, as should be closer to this side
%% Find possible bubble location
if ~isempty(pixBubSearchRange) && length(pixBubSearchRange)>3
    % If a valid range and enough to do a find peaks call
    [~,allPossibleBubbleIdx] = findpeaks(-smooth(double(...
        PSRowData.rowValOfInterest(pixBubSearchRange-int64(PSUsrInput.pinsCenter_x)))) );
    possibleBubbleIdx = allPossibleBubbleIdx(PSRowData.rowValOfInterest(...
        pixBubSearchRange(allPossibleBubbleIdx)-int64(PSUsrInput.pinsCenter_x)) ...
        <= PSRowData.bubbleDarknessFrac*255);
elseif ~isempty(pixBubSearchRange)
    % Else if there is still a range to find the bubble but not enough to call findpeaks
    possibleBubbleIdx=dsearchn(...
        smooth(double(PSRowData.rowValOfInterest(...
            pixBubSearchRange-int64(PSUsrInput.pinsCenter_x)))), ...
        min(smooth(double(PSRowData.rowValOfInterest(...
            pixBubSearchRange-int64(PSUsrInput.pinsCenter_x))))));
else
    % Else no search range and therefore no bubble in valid location
    possibleBubbleIdx=[];
end
%% Pick which possible bubble loc is bubble
```

```
if ~isempty(possibleBubbleIdx) %...
   % Crop off solid bodyies (pins, staw, etc.)
   possibleBubbleIdx_NonSolidBody = find(PSLoopingParam.frame(PSRowData.frameCol, ...
      pixBubSearchRange(possibleBubbleIdx)) > PSConditions.SolidBodyThreshold);
   if isempty(possibleBubbleIdx_NonSolidBody)
      %fprintf('No bubble points found after solid bodies removed...\n')
      % Likely no more bubble
      possibleBubLoc=possibleBubbleIdx(end) + PSRowData.bubbleStartLoc;
   else
      pixThreshedBubble_black_NonSolidBody_Pts2Use=pixBubSearchRange(...
         possibleBubbleIdx(possibleBubbleIdx_NonSolidBody));
      possibleBubLoc=double(pixThreshedBubble_black_NonSolidBody_Pts2Use(end));
   end
   detectedBubBrightness=PSRowData.rowValOfInterest(int64(dsearchn(...
      PSRowData.ipixLocOfInterest, possibleBubLoc)));
end
% Select what to do with the right most pixel found
if ~PSRowData.autoHitEdge && PSLoopingParam.iCurrentFrame>1 ...
   && (isempty(possibleBubbleIdx) ... % If couldn't find bubble
     || (nnz(PSRowData.autoBubbleloc)>1 ...
        && ( (abs(possibleBubLoc - PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame-1)) >= PSLoopingParam
            .unPhysicalPixJump) ... % Was tracking but bubble moved ahead rapidly
           || ( PSLoopingParam.iCurrentFrame > PSRowData.flameKernalDevelFrameNumb && abs(possibleBubLoc -
              PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1)) <= PSLoopingParam.unPhysicalPixJump)
              )) ... % Or bubble slipped to near recent flame loc
     || (PSLoopingParam.BubblePopedStartedFrame && ... % Or Bubble popping started and didn't really find
         bubble (row value too great/small) and not false from catching solid body (straw)
        (PSLoopingParam.iCurrentFrame <= PSLoopingParam.BubblePopedStartedFrame+5 && detectedBubBrightness >
            PSConditions.SolidBodyThreshold ...
           && abs(detectedBubBrightness-PSRowData.bubbleDarknessFrac*255)>PSRowData.bubBrightnessTolerance)
              )...
     || (possibleBubbleIdx(end)==length(pixBubSearchRange)) ) % If bubble reached max of range, bubble likley
         not here
   % Test for flame existence before hitting right edge (and after for a smooth flame)
   PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame)=PSUsrInput.pinsCenter_x;
      %if there are no black pixels in between the pins and right wall, assign x at the pins
   if ~PSRowData.BubblePopedFrame && nnz(PSRowData.autoBubbleloc) > 0
      % If hasn't yet ben detected as popped, and there was a bubble, then it must have popped here
      PSRowData.BubblePopedFrame = PSLoopingParam.iCurrentFrame;
   end
elseif PSRowData.autoHitEdge
   % If previously hit right edge...
      PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame)=PSUsrInput.pinsCenter_x;
else
   try
      PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame)=possibleBubLoc;
         % bubble location is the last black pixel before the right edge
   catch e
      fprintf('Warning no possibleBubLoc? Errored on setting possible BubLoc in getPSRowBubbleLoc\n')
   end
end

idxAutoBubLocBrightness = dsearchn(PSRowData.ipixLocOfInterest_Smoothed, ...
   PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame));
PSRowData.autoBubblelocBrightNess(PSLoopingParam.iCurrentFrame) = ...
   PSRowData.rowValOfInterest_Smoothed(idxAutoBubLocBrightness);
```

Get Tank ϕ From Partial Pressure: getPSTankEqR

```
function phi_calc = getPSTankEqR(tankTup, tankDate)
```

```
% Calculates the equivalence ratio of CH4-Air with ambient pressure offset

PGaugeOffset_psi=0;
ambPress_psi=getPSAmbPress(tankDate);

%% Calculate Fuel/Air Molar Ratio
FA_molar=(tankTup(2)-tankTup(1))/...
    (tankTup(3) + ambPress_psi+PGaugeOffset_psi - (tankTup(2)-tankTup(1)));
    % Ambient cancels in numerator
FA_molar_Stoich=1/(2*4.76); % Fuel/air
    % Methane is assumed

%% Calculate Fuel/Air Mass Ratio
presAir1 = tankTup(1)+ ambPress_psi+PGaugeOffset_psi;
presCH4 = tankTup(2)+ ambPress_psi+PGaugeOffset_psi;
presAir2 = tankTup(3)+ ambPress_psi+PGaugeOffset_psi;

MMFuel=16;
MMAir=.21*32+.79*28;
FA = MMFuel*(presCH4-presAir1) ...
    / (MMAir*(presAir1+(presAir2-presCH4)) );
stoichFA = MMFuel/(MMAir*(1+4/4-0/2)*(1+.79/.21));

%% Calculate Equivalence Ratio -- Using Molar?
phi_calc_Molar=FA_molar/FA_molar_Stoich;
phi_calc=FA/stoichFA;
```

Get Flame Center: getPSAparentFlameCenter

```
function [xCirCenter, yCirCenter, CirRad] = getPSAparentFlameCenter(PSRunData, ...
    PSUsrInput, PSLoopingParam)
% Container for getThreePtCenterCircle to get aparent center of PSRunData Points
% and put them in PSRunData at the current frame
% Updates to come w/ minimization prob to find min geo distance

% Find which index have started flaming and aren't at the pin center (reset) nor hit edge
iAdd=1;
for iRow=PSRunData.j_frameCol_pinCenter-PSLoopingParam.numRowPrior:...
        PSRunData.j_frameCol_pinCenter+PSLoopingParam.numRowPost
    if PSRunData.PSRows(iRow).isFlameStart && ~PSRunData.PSRows(iRow).autoHitEdge...
            && PSRunData.PSRows(iRow).autoflameloc(PSLoopingParam.iCurrentFrame) ...
                ~= PSUsrInput.pinsCenter_x ...
            && ~PSRunData.PSRows(iRow).isRowSuspectAtFrame(PSLoopingParam.iCurrentFrame)
        idxStartedRows(iAdd) = iRow;
        x_points(iAdd, 1) = PSRunData.PSRows(iRow).autoflameloc(PSLoopingParam.iCurrentFrame);
        y_points(iAdd, 1) = double(PSRunData.PSRows(iRow).frameCol);
        iAdd=iAdd+1;
    end
end

if ~exist('idxStartedRows','var') || length(idxStartedRows) < 3
    % If didn't find enough points that have started, just output 1D Approach...
    % x1=PSRunData.PriorPSRow.autoflameloc(PSLoopingParam.iCurrentFrame);
    % x2=PSRunData.leadPSRow.autoflameloc(PSLoopingParam.iCurrentFrame);
    % x3=PSRunData.PostPSRow.autoflameloc(PSLoopingParam.iCurrentFrame);
    %
    % y1=double(PSRunData.PriorPSRow.frameCol);
    % y2=double(PSRunData.leadPSRow.frameCol);
    % y3=double(PSRunData.PostPSRow.frameCol);
    %
    % [xCirCenter, yCirCenter] = getThreePtCenterCircle(x1, y1, x2, y2, x3, y3);
    xCirCenter = nan;
    yCirCenter = nan;
    CirRad = nan;
    return;
end

%% Best Fit Circle Center
[xCirCenter, yCirCenter, CirRad] = getPSBestFitCircle(x_points, y_points);
```

Get Best Fit Circle: getPSBestFitCircle

```
function [center_x, center_y, r] = getPSBestFitCircle(x, y)
% Calculates the circle center and radius of a best fit circle through all the points
% specified in (x,y) of given input

%% Form B, matrix
B = [x.^2+y.^2, x, y, ones(length(x),1)];

%% Compute Singular value of B
[U, S, V] = svd(B);
u_algebraic = V(:, 4);

%% deconstruct u
a=u_algebraic(1);
b=u_algebraic(2:3);
c=u_algebraic(4);

%% Determine center and radius, assuming norm(b)^2/(4*a^2) - c/a is positive
if norm(b)^2/(4*a^2)-c/a > 0
    center_x_algebraic=-b(1)/(2*a);
    center_y_algebraic=-b(2)/(2*a);
    r_algebraic=sqrt( norm(b)^2/(4*a^2) - c/a );
end
u_algebraic=[center_x_algebraic; center_y_algebraic; r_algebraic];
    % Column vector of algerbraic best fit circle as initial guest of best fit circle
% F_eval=B*u;

%% Iterate starting with min algebraic dist to find minimum geometric distance
% util=[center_x, center_y, r];
util=u_algebraic;
iter=1;
er(iter)=1; % get loop started
while er(iter) > 1E-5 && iter <=100
    % Form Jacobian
    for curPt=1:length(x)
        du(curPt,1) = norm([util(1)-x(curPt), util(2)-y(curPt)],2)-util(3);
        J_util(curPt, :) = [ ...
            (util(1)-x(curPt)) /sqrt((util(1)-x(curPt))^2+(util(2)-y(curPt))^2), ...
            (util(2)-y(curPt)) /sqrt((util(1)-x(curPt))^2+(util(2)-y(curPt))^2), ...
            -1 ];
    end
    %% Solve System of equations for correction term
    % Matlab's built in uses qr: h_Ucorrection=-J_util\du
    % Solve J(util)*h == -d(u)
    [Q, R] = qr(J_util); % Check: Q*R-J_util ~ 0;
    % QR*h=-du, R*h = -Q'*du
    h_Ucorrection=R\(-Q'*du);
    %% Check solving algorithm error of approximation:
    util=util+h_Ucorrection;
    er(iter+1)= norm(h_Ucorrection, inf) / norm(util, inf); % Should be zero...
    iter=iter+1;
end
%% Save output
center_x=util(1);
center_y=util(2);
r=util(3);
```

Function: initPSRowData

```
function PSRowData = initPSRowData(j_frameCol, PSConditions, PSUsrInput, ...
    PSLoopingParam, PSInitialFrame)
% Initialize values for PSRowData before loop
persistent initLeadPinRow

PSRowData.isFlameStart = false;
PSRowData.flameKernalDevelFrameNumb = ...
    PSLoopingParam.initflameKernalDevelFrameNumb;
    % Set an inital value for detecting initial flame kernal
    % reset to 20 frames after flame detected

% Flame Tracking
PSRowData.dumbflameloc=zeros(PSConditions.numFrames,1);
PSRowData.autoflameloc= PSUsrInput.pinsCenter_x.*ones(PSConditions.numFrames,1);
PSRowData.autoFlameLocBrightNess = zeros(PSConditions.numFrames,1);
    % Brightness values at the autoflameloc

% Bubble Tracking
PSRowData.autoBubbleloc = zeros(PSConditions.numFrames,1);
    % 0 means not processed, at pins center if errored or no bubble/popped
PSRowData.autoBubblelocBrightNess = zeros(PSConditions.numFrames,1);
    % Brightness values at the autoBubbleloc
PSRowData.BubblePopedFrame=PSLoopingParam.BubblePopedStartedFrame;
    % 0 until bubble popped, then contains frame row detected bubble pop
    % But if bubble already popped, then use

PSRowData.frameCol = j_frameCol;
PSRowData.StretchRad = nan.*zeros(PSConditions.numFrames,1);
PSRowData.possiblySparked = 0;
    % Two step to find spark...
    % First detection sets this flag to what would have been the initial spark,
    % If called again then it truely did spark...
PSRowData.misFire = false;
PSRowData.isLead = false;
    % Keep track if main lead flame

PSRowData.ipixLocOfInterest=double(0);
PSRowData.ipixLocOfInterest_Smoothed=double(0);
PSRowData.rowValOfInterest=uint8(0);
PSRowData.rowValOfInterest_Smoothed=double(0);
PSRowData.hitEdge=false;
PSRowData.autoHitEdge=false;
PSRowData.frameVidEnd=PSLoopingParam.frameVidEnd;
PSRowData.isJumpedSoModAdv=0;
PSRowData.autoPickedPixelslocFull=double(0);
PSRowData.nearestDetectedIdx=0;
PSRowData.testVectEndIdx=0;

% Need to increase number of points for brightness catching
PSRowData.rowValOfInterest_SmoothedMulFact=10;
PSRowData.startIdxPt(1) = 1;
    % The index of rowValOfInterest and ipixLocOfInterest the flame is sitting on
PSRowData.initUnPhysPixJump = 0;
    % Initial unphysical pixel jump, moved to getPSLooping ParamUpdateRow

PSRowData.NumFrameLostFlame = 0;
    % In case lose track of flame, counts the nubmer of times lost flame

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% Different Rows may have different thresholds due to Schlieren differences
% Grab pixel loc of end of Row. Tranpose to get in column major format
initRowVals=...
    transpose(PSInitialFrame(PSRowData.frameCol, ...
    int64(PSUsrInput.pinsCenter_x:size(PSInitialFrame, 2))));
initiPixVals=transpose(PSUsrInput.pinsCenter_x:size(PSInitialFrame, 2));
initSolidCutoffiPixVals=initiPixVals(initRowVals>PSConditions.SolidBodyThreshold);
PSRowData.iPixEndOfRow=initSolidCutoffiPixVals(end);

if PSConditions.isBubble
    % If Bubble, cutoff bubble from initial mean values
    PSRowData.initialMean = getPSRowMean(PSRowData.frameCol, ...
        PSConditions, PSUsrInput, PSInitialFrame);
    PSRowData.bubbleDarknessFrac=mean([ones(1,2).*PSRowData.initialMean, ...
        ones(1,3).*PSConditions.bubbleCutoffThresh]+10)/255;
    PSRowData.bubBrightnessTolerance=...
        abs(PSRowData.initialMean-PSRowData.bubbleDarknessFrac*255)/4;
        %Tolerance bubble can be found before deemed not found
    PSRowData.bubbleStartLoc=0; % Start pixel search point for bubble location

    % End pixel search point for bubble location
    if isfield('PSUsrInput', 'pixel_bubble_right_auto')
        PSRowData.bubbleEndSearchLoc=...
            mean([ones(1,3).*PSUsrInput.pixel_bubble_right_auto, PSUsrInput.pixEdgeOfRIO]);
    else % Use usr input if have to when initializing pixel_bubble_right_auto
        PSRowData.bubbleEndSearchLoc=...
            mean([ones(1,3).*PSUsrInput.pixel_bubble_right_avg, PSUsrInput.pixEdgeOfRIO]);
    end
    % Same as above but for pin center...do once
    if PSLoopingParam.iCurrentFrame==1
        % If initial frame, reset initLeadPinRow since isempty call only
        % checks for existence and will use other runs for this value...
        initLeadPinRow=getPSRowMean(int64(PSUsrInput.pinsCenter_y), ...
            PSConditions, PSUsrInput, PSInitialFrame);
    end
    brightDevFromLead=(PSRowData.initialMean-initLeadPinRow);
    if brightDevFromLead>0
        % Only add if brighter to better catch flame in row that is bias brighter
        PSRowData.flameCutoffThreshold=...
            PSConditions.flameCutoffThreshold + brightDevFromLead;
    else
        PSRowData.flameCutoffThreshold=PSConditions.flameCutoffThreshold;
    end
else
    % Else solid body cutoff should be good enough to estimate mean brightness value
    PSRowData.initialMean=getPSRowMean(PSRowData.frameCol,...
        PSConditions, PSUsrInput, PSInitialFrame);
    PSRowData.flameCutoffThreshold=PSConditions.flameCutoffThreshold;
end

PSRowData.isRowSuspectAtFrame = logical(zeros(PSConditions.numFrames,1));
```

Function: interpPSRunData

```
function PSPostProcData = interpPSRunData(PSRunData, PSScopeData, PSConditions, PSUsrInput, PSPlotDispProp,
     PSFlags)
% Interpret PSRunData, applying filters to radius and splining, generating main plotting parameters

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Filtering Radius Data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Savitzky-Golay Filtering
PSPostProcData.SG_order=3;

% calculate 3% range for framelength SG smoothing
sg_framlenThreePer=length(PSRunData.r_cm_raw_cropped)*3/100;
if ~mod(sg_framlenThreePer,2) % if even, add one. Requires odd number
    sg_framlenThreePer=sg_framlenThreePer+1;
end
if PSConditions.isBubble
    PSPostProcData.SG_framelen=9; %Sets SG Framelength for rest of code,
        % increase for more smoothing, compare to sg_framlenThreePer
        % ~9 for Stratified/Bubble
else
    PSPostProcData.SG_framelen=15;
        % ~15 for Homogenous
end
PSPostProcData.r_cm_SG = sgolayfilt(PSRunData.r_cm_raw_cropped,PSPostProcData.SG_order, ...
    PSPostProcData.SG_framelen);
PSPostProcData.r_cm_SG_Aparent = sgolayfilt(PSRunData.r_cm_raw_Aparent_cropped, ...
    PSPostProcData.SG_order, PSPostProcData.SG_framelen);
    % Aparent for Stretch Rate Radius Calc
PSPostProcData.r_cm_SG_Aparent_avg = sgolayfilt(PSRunData.r_cm_raw_Aparent_cropped_avg, ...
    PSPostProcData.SG_order, PSPostProcData.SG_framelen);


if ~PSFlags.suppressDebugOutput
    fprintf(['Using a %d Order Savitzky-Golay Filter with frame length of %d',...
        ' (%.2f corresponds to 3%%)\n'], PSPostProcData.SG_order, ...
        PSPostProcData.SG_framelen, sg_framlenThreePer);
end
% Median Filtering - Not used
Med_order=4;
r_cm_Med = medfilt1(PSRunData.r_cm_raw_cropped-PSRunData.r_cm_raw_cropped(end),Med_order) +...
    PSRunData.r_cm_raw_cropped(end);

%Spline
if length(PSRunData.timeReal_raw_cropped) ~= length(PSRunData.timeReal_interp)
    PSPostProcData.r_cm_interp_spline = ...
        spline(PSRunData.timeReal_raw_cropped, PSRunData.r_cm_raw_cropped, ...
        PSRunData.timeReal_interp);
    PSPostProcData.r_cm_interp_SG = ppval(spline(PSRunData.timeReal_raw_cropped, ...
        PSPostProcData.r_cm_SG), PSRunData.timeReal_interp);
        %Using S-G Filtering and spline
    PSPostProcData.r_cm_interp_Med = ppval(spline(PSRunData.timeReal_raw_cropped, ...
        r_cm_Med), PSRunData.timeReal_interp);
else
    PSPostProcData.r_cm_interp_spline = PSRunData.r_cm_raw_cropped;
    PSPostProcData.r_cm_interp_SG = PSPostProcData.r_cm_SG;
    PSPostProcData.r_cm_interp_Med = r_cm_Med;
end
```

```
if ~PSFlags.isFilterRadius
   PSPostProcData.r_cm_interp = PSPostProcData.r_cm_interp_spline;
      % Chosen interpolated result used in flame speed, etc.
else
   PSPostProcData.r_cm_interp = PSPostProcData.r_cm_interp_SG; % Smooths more...
end

% Determine Aparent Radius from stretch rate radius, spline from cropped value
PSPostProcData.r_cm_SG_Aparent(isnan(PSPostProcData.r_cm_SG_Aparent))=0;
PSPostProcData.r_cm_SG_Aparent(abs(PSPostProcData.r_cm_SG_Aparent)>10*1024)=0;
   % Set non-sensical values to 0 radius
try
   PSPostProcData.aparentRadStart = ...
      find(PSPostProcData.r_cm_SG_Aparent(1:...
         int64(length(PSPostProcData.r_cm_SG_Aparent)/4))<=0, 1, 'last') + 1;
         % Grab point where radius makes sense after it has been developed
   PSPostProcData.aparentRadEnd = length(PSPostProcData.r_cm_SG_Aparent);
   PSPostProcData.r_cm_interp_SG_Aparent = ppval(spline(...
      PSRunData.timeReal_raw_cropped(PSPostProcData.aparentRadStart:...
         PSPostProcData.aparentRadEnd), ...
      PSPostProcData.r_cm_SG_Aparent(PSPostProcData.aparentRadStart:...
         PSPostProcData.aparentRadEnd)), ...
      PSRunData.timeReal_interp(PSPostProcData.aparentRadStart:end) );
       % Aparent for Stretch Rate Radius Calc
catch e % If errored in finding the aparentstart position
   PSPostProcData.aparentRadStart=...
      length(PSRunData.timeReal_interp)-PSPostProcData.aparentRadEnd+1;
      % Needs to be this size for later plotting, etc.
   PSPostProcData.r_cm_interp_SG_Aparent = ...
      zeros(size(PSRunData.timeReal_interp(PSPostProcData.aparentRadStart:end)));
      % Would have been interpolated to this size
   fprintf('Warning: PSPostProcData.r_cm_interp_SG_Aparent set to 0s\n')
end

% Doing averaged flame center aparent flame radius:
if sum(~isnan(PSPostProcData.r_cm_SG_Aparent_avg)) &&...
      sum(PSPostProcData.r_cm_SG_Aparent_avg) && ...
      sum(abs(PSPostProcData.r_cm_SG_Aparent_avg)./...
         length(PSPostProcData.r_cm_SG_Aparent_avg)) < 5
   % If there is at least 1 data point...and values make sense
   PSPostProcData.r_cm_interp_Aparent_avg = ...
      spline(PSRunData.timeReal_raw_cropped, PSPostProcData.r_cm_SG_Aparent_avg, ...
      PSRunData.timeReal_interp);
else
   PSPostProcData.r_cm_interp_Aparent_avg = zeros(length(PSRunData.timeReal_interp),1);
end

% Must be strictly increasing at begining, crop off those values
PSPostProcData.aparentStartIdx=1;
for i=2:10 % Look at the first 10 points for a decreasing term. set to first increasing value
   if PSPostProcData.r_cm_interp_Aparent_avg(PSPostProcData.aparentStartIdx) >= ...
         PSPostProcData.r_cm_interp_Aparent_avg(i)
      % If decreasing, crop this point off.. must be just starting off
      PSPostProcData.aparentStartIdx=i;
   end
end

% Find location of bubble, index
if PSConditions.isBubble
   [PSPostProcData.initialBubble_cm_interpLocIndex] = ...
```

```
        dsearchn(PSPostProcData.r_cm_interp, PSUsrInput.pinCentRadialBubbleLoc_Auto_cm);
else
    % Just pick a centerpoint if no bubble
    PSPostProcData.initialBubble_cm_interpLocIndex=...
        int64( (PSRunData.startSpotPastFlameKernal+length(PSRunData.timeReal_interp))/2 );
end

%% Find slope of lines, internal/(Transition)/external to bubble
PSPostProcData.idx_InternalBubbleZone_start=int64(PSRunData.startSpotPastFlameKernal);
    % =40 for flame wripple in 20181026_121419, PSRunData.startSpotPastFlameKernal;
PSPostProcData.idx_InternalBubbleZone_end=PSPostProcData.initialBubble_cm_interpLocIndex;

% Internal to Bubble Zone
PSPostProcData.time_InternalBubbleZoneInterp_ms=[...
    ones(length(PSRunData.timeReal_interp(PSPostProcData.idx_InternalBubbleZone_start:...
        PSPostProcData.idx_InternalBubbleZone_end)),1),...
    PSRunData.timeReal_interp(PSPostProcData.idx_InternalBubbleZone_start:...
        PSPostProcData.idx_InternalBubbleZone_end).*1000];
PSPostProcData.rad_InternalBubbleZone=...
    PSPostProcData.r_cm_interp(PSPostProcData.idx_InternalBubbleZone_start:...
        PSPostProcData.idx_InternalBubbleZone_end);
rad_InternalBubble_linefit=...
    PSPostProcData.time_InternalBubbleZoneInterp_ms\PSPostProcData.rad_InternalBubbleZone;
PSPostProcData.rad_InternalBubble_slope_cms=rad_InternalBubble_linefit(2)*1000;

if PSConditions.isBubble
    % Transition to Bubble Zone
    PSPostProcData.idx_TransitionBubbleZone_start=PSPostProcData.idx_InternalBubbleZone_end;

    % Find end of transition
    rTransitionAddition_cm=2*(1.8-1.5);
    PSPostProcData.idx_TransitionBubbleZone_end= ...
        dsearchn(PSPostProcData.r_cm_interp, ...
            PSUsrInput.pinCentRadialBubbleLoc_Auto_cm+rTransitionAddition_cm);

    PSPostProcData.time_TransitionBubbleZoneInterp_ms=[...
        ones(length(PSRunData.timeReal_interp(PSPostProcData.idx_TransitionBubbleZone_start:...
            PSPostProcData.idx_TransitionBubbleZone_end)),1),...
        PSRunData.timeReal_interp(PSPostProcData.idx_TransitionBubbleZone_start:...
            PSPostProcData.idx_TransitionBubbleZone_end).*1000];
    PSPostProcData.rad_TransitionBubbleZone=...
        PSPostProcData.r_cm_interp(PSPostProcData.idx_TransitionBubbleZone_start:...
            PSPostProcData.idx_TransitionBubbleZone_end);
    rad_TransitionBubble_linefit=PSPostProcData.time_TransitionBubbleZoneInterp_ms\...
        PSPostProcData.rad_TransitionBubbleZone;
    PSPostProcData.rad_TransitionBubble_slope_cms=rad_TransitionBubble_linefit(2)*1000;

    % External to Bubble Zone Start
    PSPostProcData.idx_ExternalBubbleZone_start=PSPostProcData.idx_TransitionBubbleZone_end;
else
    % External to Bubble Zone Start
    PSPostProcData.idx_ExternalBubbleZone_start=PSPostProcData.idx_InternalBubbleZone_end;
end

% External to Bubble Zone
PSPostProcData.idx_ExternalBubbleZone_end=length(PSRunData.timeReal_interp);

PSPostProcData.time_ExternalBubbleZoneInterp_ms=[...
    ones(length(PSRunData.timeReal_interp(PSPostProcData.idx_ExternalBubbleZone_start:...
        PSPostProcData.idx_ExternalBubbleZone_end)),1),...
    PSRunData.timeReal_interp(PSPostProcData.idx_ExternalBubbleZone_start:...
        PSPostProcData.idx_ExternalBubbleZone_end).*1000];
PSPostProcData.rad_ExternalBubbleZone=...
    PSPostProcData.r_cm_interp(PSPostProcData.idx_ExternalBubbleZone_start:...
        PSPostProcData.idx_ExternalBubbleZone_end);
rad_ExternalBubble_linefit=PSPostProcData.time_ExternalBubbleZoneInterp_ms\...
    PSPostProcData.rad_ExternalBubbleZone;
PSPostProcData.rad_ExternalBubble_slope_cms=rad_ExternalBubble_linefit(2)*1000;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Radius Spline Sanity Check Plot
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if PSFlags.dispKeyPlots || PSFlags.savePostPlots
    figHandSpline=figure('Visible', 'On'); % Spline Check
    if ~PSFlags.dispKeyPlots
        figHandSpline.Visible='off';
    end
    plot(PSRunData.timeReal_raw_cropped*1000,PSRunData.r_cm_raw_cropped,...
        'o', 'DisplayName', 'Raw');
    hold on
    plot(PSRunData.timeReal_interp*1000,PSPostProcData.r_cm_interp_spline, ...
        'DisplayName', 'Spline');
    plot(PSRunData.timeReal_interp*1000,PSPostProcData.r_cm_interp_SG, ...
        'DisplayName', 'Spline Savitzky-Golay');
    plot(PSRunData.timeReal_interp(PSPostProcData.aparentStartIdx:end)*1000, ...
        PSPostProcData.r_cm_interp_Aparent_avg(PSPostProcData.aparentStartIdx:end), ...
        'g', 'DisplayName', 'Aparent Center')
    % plot(PSRunData.timeReal_interp*1000,PSPostProcData.r_cm_interp_Med, 'c')
    xlabel('Time [ms]'), ylabel('r [cm]');
    title(strcat('Raw compared with Spline: cm vs. ms data (', ...
        PSConditions.VideoName, ')' ) , 'Interpreter','none')
    legend('location', 'best'), legend show

    if PSConditions.isBubble
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Find and add bubble location in specified thickness of bubble
        ind_bubbleStart_SG = dsearchn(PSPostProcData.r_cm_interp_SG, ...
            PSUsrInput.pinCentRadialBubbleLoc_Auto_cm);
        ind_bubbleEnd_SG=min(ind_bubbleStart_SG+1, length(PSPostProcData.r_cm_interp_SG));
        r_x=PSRunData.timeReal_interp(ind_bubbleStart_SG)*1000;
        r_y=min(PSPostProcData.r_cm_interp_SG);
        r_dx=PSRunData.timeReal_interp(ind_bubbleEnd_SG)*1000-...
            PSRunData.timeReal_interp(ind_bubbleStart_SG)*1000;
        r_dy=(max(PSPostProcData.r_cm_interp_SG)-min(PSPostProcData.r_cm_interp_SG));
        R=rectangle('Position', [r_x, r_y, r_dx, r_dy],'Tag', 'BubbleLocation',...
            'FaceColor', [0.302 0.745 0.933]); %, 'EddgeColor','r')
        hold off; alpha(.25);
        if PSFlags.debug_autoScaleAxis
            text(r_x+2*r_dx, r_y+r_dy*9/10, sprintf('Initial\nBubble\nLocation'))
        else
            text(r_x+2*r_dx, r_y+r_dy*9/10, sprintf('Initial\nBubble\nLocation'))
        end
        hold off; alpha(.5)

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Find location the bubble popped
        PSPostProcData.bubblePopLocation = ...
            PSRunData.leadPSRow.autoBubbleloc(PSRunData.leadPSRow.BubblePopedFrame-1) -...
            PSUsrInput.pinsCenter_x;
        PSPostProcData.bubblePopLocation_cm = ...
```

```
            PSPostProcData.bubblePopLocation/PSUsrInput.pixel2cm;
        PSPostProcData.bubblePopLocation_Aparent_avg = ...
            PSRunData.leadPSRow.autoBubbleloc(PSRunData.leadPSRow.BubblePopedFrame-1) -...
            PSRunData.x_Aparent_avg;
        PSPostProcData.bubblePopLocation_Aparent_avg_cm = ...
            PSPostProcData.bubblePopLocation_Aparent_avg/PSUsrInput.pixel2cm;
        ind_bubblePop_SG = dsearchn(PSPostProcData.r_cm_interp_SG, ...
            PSPostProcData.bubblePopLocation_cm);
        ind_bubblePop_Aparent_avg = dsearchn(PSPostProcData.r_cm_interp_Aparent_avg, ...
            PSPostProcData.bubblePopLocation_Aparent_avg_cm);
        hold on;
        h=plot(PSRunData.timeReal_interp(ind_bubblePop_SG)*1000,...
            PSPostProcData.r_cm_interp_SG(ind_bubblePop_SG),'xb');
            %, 'DisplayName', 'Spline Savitzky-Golay');
        h.Annotation.LegendInformation.IconDisplayStyle='off';
        h=plot(PSRunData.timeReal_interp(ind_bubblePop_Aparent_avg)*1000, ...
            PSPostProcData.r_cm_interp_Aparent_avg(ind_bubblePop_Aparent_avg), 'xb');
            %, 'DisplayName', 'Aparent Center')
        h.Annotation.LegendInformation.IconDisplayStyle='off';
        hold off;
    end
    filename =['Raw_Spline_GS_Flame_Radius_vs_Time_',...
        PSConditions.rawVideofilename,'.eps']; %Name of file to be saved
    outputVideoPath=fullfile(PSConditions.folderPath, filename);
    if PSFlags.savePostPlots
        saveas(figHandSpline, outputVideoPath, 'epsc')
    end
    if strcmp(figHandSpline.Visible, 'off')
        close(figHandSpline)
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Calculate the velocity data cm/s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PSPostProcData.drdt_f=zeros(length(PSRunData.timeReal_interp)-1, 1);
    %Forward diff
PSPostProcData.drdt_c = zeros(length(PSRunData.timeReal_interp)-1, 1);
    %Center diff
PSPostProcData.drdt_5pt = zeros(length(PSRunData.timeReal_interp)-1, 1);
    % 5 point center diff O(h^4)
PSPostProcData.drdt_5pt_SG = zeros(length(PSRunData.timeReal_interp)-1, 1);
PSPostProcData.drdt_5pt_Med = zeros(length(PSRunData.timeReal_interp)-1, 1);

% DrDt = getPSRadiusVelocity(time, radius)
PSPostProcData.drdt_5pt = getPSRadiusVelocity(PSRunData.timeReal_interp, ...
    PSPostProcData.r_cm_interp);
    % 5 point center diff O(h^4)
PSPostProcData.drdt_5pt_SG = getPSRadiusVelocity(PSRunData.timeReal_interp, ...
    PSPostProcData.r_cm_interp_SG);
PSPostProcData.drdt_5pt_Med = getPSRadiusVelocity(PSRunData.timeReal_interp, ...
    PSPostProcData.r_cm_interp_Med);

%Choose O(h^4) center diff
PSPostProcData.drdt=PSPostProcData.drdt_5pt_SG;
PSPostProcData.drdt_mean=...
    mean(PSPostProcData.drdt_5pt(PSRunData.startSpotPastFlameKernal:end));
    % Move to cutoff initial flame peak due to flame kernal development
PSPostProcData.drdt_mean_SG=...
    mean(PSPostProcData.drdt_5pt_SG(PSRunData.startSpotPastFlameKernal:end));
    % Move to cutoff initial flame peak due to flame kernal development
```

```
if PSConditions.isBubble
    PSPostProcData.drdt_beforeBubble=...
        mean(PSPostProcData.drdt_5pt(PSRunData.startSpotPastFlameKernal:...
            PSPostProcData.initialBubble_cm_interpLocIndex));
    PSPostProcData.drdt_afterBubble=...
        mean(PSPostProcData.drdt_5pt(PSPostProcData.initialBubble_cm_interpLocIndex:end));
else
    PSPostProcData.drdt_beforeBubble=PSPostProcData.drdt_mean;
    PSPostProcData.drdt_afterBubble=PSPostProcData.drdt_mean;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Stretch Rate calcs...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate the stretch rate 1/s, assuming burned and unbruned have same density
isUseNonLinearExtrap=true;
    % Whether to use a linear method to extrapolate to zero stretch,
    % otherwise uses a more robust non-linear method
if true % For code folding...
    %% Find T and P for density comensation, Assuming equal pressures
    [PSPostProcData.TBurnt, PSPostProcData.SLoSimulated, r_cm_SR_rStart, r_cm_SR_rEnd] = ...
        getPSTBurntNSLo(PSConditions.phiPost);
    PSPostProcData.TUnburnt=PSScopeData.tempAutoDetected +273.15;
        % K, Ti from Text File measuring gas unburnt temp, ~ambientTemp
    mmBurnt=27.62; mmUnburnt=27.62;
    %% Initialize Variables. nan so doesn't plot values that don't exist
    PSPostProcData.SR_SG = nan.*zeros(length(PSPostProcData.drdt),1);
    PSPostProcData.SRMod_SG = nan.*zeros(length(PSPostProcData.drdt),1);
    PSPostProcData.SR_SG_drdt = nan.*zeros(length(PSPostProcData.drdt),1);

    %% Calculate raw stretch rate, different ways
    SR_Spherical_Fact_cm_interp_SG=2./PSPostProcData.r_cm_interp_SG(1:end-1);

    PSPostProcData.SR_SG_drdt(1:length(SR_Spherical_Fact_cm_interp_SG)) = ...
        PSPostProcData.drdt(1:length(SR_Spherical_Fact_cm_interp_SG)).*...
            SR_Spherical_Fact_cm_interp_SG;

    % Separate loop for aparent as cutting off initial values when radius not detected
    for j = PSPostProcData.aparentRadStart:...
            min(length(PSPostProcData.r_cm_interp_SG_Aparent), length(PSPostProcData.drdt))
        PSPostProcData.SRMod_SG_drdt(j) = ...
            PSPostProcData.drdt(j)*2/ (PSPostProcData.r_cm_interp_SG_Aparent(j));
    end

    %% Crop and Smooth Stretch rate values, based on end radius given by getPSTBurntNSLo
    PSPostProcData.SR_SG_rStart= max(dsearchn(...
        PSPostProcData.r_cm_interp_SG, r_cm_SR_rStart), 1);
        % Crop off, so only looking at stretch rate from 1.0 to 2.5 cm....

    % If using regulat SR calculated from drdt...Literature way
    PSPostProcData.SR_SG_rEnd = min(dsearchn(PSPostProcData.r_cm_interp_SG, r_cm_SR_rEnd), ...
        length(PSPostProcData.SR_SG_drdt));
        % Crop off, so only looking at stretch rate from 1.0 cm to 2.5 cm....

    % Smooth stretch rate to reduce noise
    PSPostProcData.SR_SG_drdt_Filtered=...
        sgolayfilt(PSPostProcData.SR_SG_drdt(PSPostProcData.SR_SG_rStart:...
            PSPostProcData.SR_SG_rEnd), ...
        PSPostProcData.SG_order, PSPostProcData.SG_framelen);
    x=PSPostProcData.SR_SG_drdt_Filtered;
```

```
%% Find Unstretched Flame Speed, linear method:
% Slope Intercept for 0 stretch laminar flame speed w/o compensating for density yet
y=PSPostProcData.drdt(PSPostProcData.SR_SG_rStart:PSPostProcData.SR_SG_rEnd);
X=[ones(length(x),1), x];
PSPostProcData.SL_linefit=X\y;
PSPostProcData.SLo_linear=PSPostProcData.SL_linefit(1);
    % Slope intersept for zero stretch extrapolated from data

%% Find Unstretched Flame Speed, non-linear method:
% ln(dr_dt) = ln(dr_dt_zeroStretch) -alpha*dr_dt_zeroStretch*Lb, alpha=2/(Rf*dr_dt)
y_two_RfSb=log(PSPostProcData.drdt(PSPostProcData.SR_SG_rStart:PSPostProcData.SR_SG_rEnd));
    % Form natural log for
x_two_RfSb=2./(PSPostProcData.r_cm_interp_SG(PSPostProcData.SR_SG_rStart:...
    PSPostProcData.SR_SG_rEnd)...
        .*PSPostProcData.drdt(PSPostProcData.SR_SG_rStart:PSPostProcData.SR_SG_rEnd));
X_rfSb=[ones(length(x_two_RfSb),1), x_two_RfSb];
nonLinearSL_linefit=X_rfSb\y_two_RfSb;
lnSbUs=nonLinearSL_linefit(1);
PSPostProcData.SLo_nonLin = exp(lnSbUs);

%% Determine Markstein length/number Lb
PSPostProcData.Lbs_linear = (PSPostProcData.drdt(PSPostProcData.SR_SG_rStart:...
    PSPostProcData.SR_SG_rEnd)-PSPostProcData.SLo_linear)...
        ./(-PSPostProcData.SR_SG_drdt_Filtered);
PSPostProcData.Lb_linear = mean(PSPostProcData.Lbs_linear);

PSPostProcData.Lbs_nonLin = (log(PSPostProcData.drdt(PSPostProcData.SR_SG_rStart:...
    PSPostProcData.SR_SG_rEnd))-log(PSPostProcData.SLo_nonLin))./...
        (-2.*PSPostProcData.SLo_nonLin./...
            (PSPostProcData.r_cm_interp_SG(PSPostProcData.SR_SG_rStart:...
                PSPostProcData.SR_SG_rEnd).*...
            PSPostProcData.drdt(PSPostProcData.SR_SG_rStart:PSPostProcData.SR_SG_rEnd)));
PSPostProcData.Lb_nonLin = mean(PSPostProcData.Lbs_nonLin);

%% Plot Markstein Lengths vs. Rf, taken the mean
if PSFlags.dispKeyPlots || PSFlags.savePostPlots
    markSteinFigHand=figure('Visible', 'On'); %plots velocity vs time
    if ~PSFlags.dispKeyPlots
        markSteinFigHand.Visible='off';
    end
    plot(PSPostProcData.r_cm_interp_SG(PSPostProcData.SR_SG_rStart:...
        PSPostProcData.SR_SG_rEnd), PSPostProcData.Lbs_linear, 'DisplayName', 'Linear')
    hold on
    plot(PSPostProcData.r_cm_interp_SG(PSPostProcData.SR_SG_rStart:...
        PSPostProcData.SR_SG_rEnd), PSPostProcData.Lbs_nonLin, 'DisplayName','Nonlinear')
    legend('show')
    ylabel('Markstein Length')
    xlabel('Rf')

    filename =['Lb_vs_Radius_',PSConditions.rawVideofilename,'.eps'];
    outputVideoPath=fullfile(PSConditions.folderPath, filename);
    if PSFlags.savePostPlots
        saveas(markSteinFigHand, outputVideoPath, 'epsc')
    end
    if strcmp(markSteinFigHand.Visible, 'off')
        close(markSteinFigHand)
    end
end

%% Compensate for density difference
```

```
    rhou_rhob=(mmUnburnt/mmBurnt)*(PSPostProcData.TBurnt/PSPostProcData.TUnburnt);

    % Linear method
    PSPostProcData.SLo_linear_denComp = PSPostProcData.SLo_linear/rhou_rhob;

    % Nonlinear method
    PSPostProcData.SLo_nonLin_denComp = PSPostProcData.SLo_nonLin/rhou_rhob;

    PSPostProcData.SR_SG=PSPostProcData.SR_SG_drdt;

    % Save relevant parameters
    if isUseNonLinearExtrap
        PSPostProcData.SLo = PSPostProcData.SLo_nonLin;
        PSPostProcData.SLo_denComp = PSPostProcData.SLo_nonLin_denComp;
    else
        PSPostProcData.SLo = PSPostProcData.SLo_linear;
        PSPostProcData.SLo_denComp = PSPostProcData.SLo_linear_denComp;
    end

    %% Write output if necessary
    % if ~PSFlags.suppressDebugOutput
    if ~PSConditions.isBubble
        % No need to print when there is a bubble... but still calculated above for now
        fprintf('\tLinear: %.2f cm/s (%.3f cm/s with rho comp) Lb=%.2f\n', ...
            PSPostProcData.SLo_linear, PSPostProcData.SLo_linear_denComp, ...
            PSPostProcData.Lb_linear)
        fprintf('\tNonlinear: %.2f cm/s (%.3f cm/s with rho comp) Lb=%.2f\n', ...
            PSPostProcData.SLo_nonLin, PSPostProcData.SLo_nonLin_denComp, ...
            PSPostProcData.Lb_nonLin)
        fprintf('\t(Simulated SLo: %.3f cm/s)\n', ...
            PSPostProcData.SLoSimulated)
        fprintf('\t(Burned Gas Temp assumed: %.2f K, Unburned Gas Temp: %.2f C)\n', ...
            PSPostProcData.TBurnt, PSPostProcData.TUnburnt-273.15)
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Radius Expansion Calculation if Lean or Air
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if PSConditions.isBubble % && PSConditions.phiPost
    fprintf('\nCalculating expansion radius of Bubble\n')

    % Left of center
    loc_cm=(PSUsrInput.pinsCenter_x-PSUsrInput.pixel_bubble_left_avg)/PSUsrInput.pixel2cm;

    % Right of Center
    roc_cm=(PSUsrInput.pixel_bubble_right_auto-PSUsrInput.pinsCenter_x)/PSUsrInput.pixel2cm;

    fprintf(['\tCenter offset, from left side %.3f cm, from right side %.3f cm',...
        ' (Bubble off pin center by %.3f cm from selected right bubble endpoint)\n'], ...
        loc_cm, roc_cm, abs((PSUsrInput.pinsCenter_x)-PSUsrInput.bubbleCenter_rad_pixLoc)/...
        PSUsrInput.pixel2cm);

    [initial_TBurnt, ~, ~, ~] = getPSTBurntNSLo(PSConditions.phiInit);
    initial_rhou_rhob=(mmUnburnt/mmBurnt)*(initial_TBurnt/PSPostProcData.TUnburnt);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Find Expected burned radius
    % assuming SPHERICAL burning from center perfectly aligned with bubble
    radiusUnburned_cm=PSUsrInput.dBubble_selected_cm/2;
```

```
expectRadiusBurnedFromPinCenter_cm = radiusUnburned_cm*nthroot(initial_rhou_rhob,3);

%% Find bubble-pin center offset
%bubbleCent_Pix=(PSUsrInput.pixel_bubble_right-PSUsrInput.pixel_bubble_left)/2;
PSUsrInput.bubblePinOffset_x_pix = ...
    PSUsrInput.bubbleCenter_x_pixLoc - PSUsrInput.pinsCenter_x;
PSUsrInput.bubblePinOffset_x_cm = PSUsrInput.bubblePinOffset_x_pix/PSUsrInput.pixel2cm;

PSUsrInput.bubblePinOffset_y_pix = ...
    PSUsrInput.bubbleCenter_y_pixLoc - PSUsrInput.pinsCenter_y;
PSUsrInput.bubblePinOffset_y_cm = PSUsrInput.bubblePinOffset_y_pix/PSUsrInput.pixel2cm;

%% Find Expected radius if bubble expanded from center of initial bubble
expectRadiusBurnedFromBubCirCenter_cm = ...
    (PSUsrInput.bubbleCenter_rad_pixLoc/PSUsrInput.pixel2cm)*nthroot(initial_rhou_rhob,3);

% Shift expected radius based on bubble circle center
expectMaxRadFromPinCenter_CorrectedforBubbleOffCenterX = ...
    expectRadiusBurnedFromBubCirCenter_cm - PSUsrInput.bubblePinOffset_x_cm;

%% Find farthest detected radius for all rows tracked
farthestPinCenRadiusDetected_cm = max(PSRunData.r_cm_raw_cropped); %(end);

farthestRadiusDetected_cm=farthestPinCenRadiusDetected_cm;
for i=1:length(PSRunData.PSRows)
    if ~isempty(PSRunData.PSRows(i).autoflameloc) && ...
            (max(PSRunData.PSRows(i).autoflameloc)-PSUsrInput.pinsCenter_x)/...
            PSUsrInput.pixel2cm>farthestRadiusDetected_cm
        farthestRadiusDetected_cm=(max(PSRunData.PSRows(i).autoflameloc)-...
            PSUsrInput.pinsCenter_x)/PSUsrInput.pixel2cm;
    end
end

%% Print radius data
fprintf(strcat('\n\tSpherical Method: Expected flame radius at end:', ...
  '\n %.3f cm if expanded at pin center,',...
  '\n %.3f cm from pin center_x if expanded from bubble center,',...
  '\n\tFarthest radius detected from raw cropped (usually pin center): %.3f cm\n'), ...
  expectRadiusBurnedFromPinCenter_cm, ...
  expectMaxRadFromPinCenter_CorrectedforBubbleOffCenterX, farthestPinCenRadiusDetected_cm)

lastRadiusExtrap_cm=PSRunData.CirRad(PSRunData.r_raw_firstEndZero-1)/PSUsrInput.pixel2cm;
fprintf('\t\tLast Extrapolated radius (circen): %.3f cm\n', lastRadiusExtrap_cm)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Calculate TORROIDAL Possible Expanded Volumes
aRadSrVol_cm=transpose(2:1E-2:5);
    % Apparent Radius to search for estimated Expanded volume

d_r=getPSChordDist(aRadSrVol_cm, PSRunData, PSUsrInput, PSFlags);
theta=acos(d_r./aRadSrVol_cm);

% Calculate Torroid Volumes depending on given radius
volToSearch_Torroid=2*pi^2.*d_r.*aRadSrVol_cm.^2 ...
    - pi.*d_r.*aRadSrVol_cm.^2.*((theta)- sin(theta));
    % Volume of Torroid, derived by hand

%% Assuming Torroidal burning from pin center with size of bubble from radius input by user
volToMatch_UserInputDia=4/3*pi*radiusUnburned_cm^3*initial_rhou_rhob;
% Determine expanded bubble volume from initial unburned radius assumed from user input
```

```
radIdxFound_UserInputDia = dsearchn(volToSearch_Torroid, volToMatch_UserInputDia);

%% Assuming TORROIDAL burning from center perfectly aligned with initial bubble center
volExpandedFromInitAutoBub=...
   4/3*pi*(PSUsrInput.bubbleCenter_rad_pixLoc/PSUsrInput.pixel2cm)^3*initial_rhou_rhob;
radIdxFound_InitAutoBub = dsearchn(volToSearch_Torroid, volExpandedFromInitAutoBub);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Quick plot to check w/ plot to solve nonlinear eqn
if PSFlags.dispKeyPlots || PSFlags.savePostPlots
   volFigHand=figure('Visible', 'On'); %plots velocity vs time
   if ~PSFlags.dispKeyPlots
      volFigHand.Visible='off';
   end
   % Plot radius as calculated from torroidal calc (Aparent radius+aparent center offset)
   plot(aRadSrVol_cm+getPSChordDist(aRadSrVol_cm, PSRunData, PSUsrInput, PSFlags),...
      volToSearch_Torroid, ...
      'k', 'DisplayName', 'Torroidal Expanded Volume')
      % Apparent Radius + distace from aparent center to pin center
   hold on
   % Plot initial volume in overbar
   plot([aRadSrVol_cm(1),aRadSrVol_cm(end)], 4/3*pi()*radiusUnburned_cm^3.*[1,1],...
      'b--', 'DisplayName', 'Initial Bubble Volume')
   % Plot expanded radiusUnburned_cm in overbar
   plot([aRadSrVol_cm(1),aRadSrVol_cm(end)], volToMatch_UserInputDia.*[1,1], ...
      'g', 'DisplayName', 'Bubble Expanded Volume to Match')
   % Plot radius to maximum radius of expanded sphere from pin center
   plot(expectRadiusBurnedFromPinCenter_cm.*[1,1], ...
      [0, max([volToSearch_Torroid(end),volToMatch_UserInputDia])],...
      'g', 'DisplayName', 'Expected Spherical Radius')
   % Plot farthest radius detected from pin center and farthest detected
   plot(farthestPinCenRadiusDetected_cm.*[1,1], ...
      [0, max([volToSearch_Torroid(end),volToMatch_UserInputDia])], ...
      '-+r', 'DisplayName', 'Furthest Radius @ pin center Detected')
   plot(farthestRadiusDetected_cm.*[1,1], ...
      [0, max([volToSearch_Torroid(end),volToMatch_UserInputDia])], ...
      'r', 'DisplayName', 'Furthest Burned Radius Detected')
   title({['Expected/Tracked Burned Volumes for ', PSConditions.VideoName], ...
      ['Radius predict/matching, stopped at: ', num2str(PSRunData.flameStopIdx), ...
         '/', num2str(PSRunData.r_raw_firstEndZero), ...
         ' (', num2str(PSConditions.numFrames), ' Total)']}, ...
      'interpreter', 'none')
   xlabel('Expanded Burnt Bubble Gas Volume End Radius', 'interpreter', 'none')
   ylabel('Volume [cm^3]', 'interpreter', 'none')
   legend('location','best', 'interpreter', 'none')
   legend show

   filename =['Expanded_Volume_vs_Radius_',PSConditions.rawVideofilename,'.eps'];
   outputVideoPath=fullfile(PSConditions.folderPath, filename);

   if PSFlags.savePostPlots
      saveas(volFigHand, outputVideoPath, 'epsc')
   end
   if strcmp(volFigHand.Visible, 'off')
      close(volFigHand)
   end
end
lastApparentRadiusDetected_cm = PSRunData.r_cm_raw_Aparent_cropped(end);
expectApparentRadiusTorroidBurned_cm=aRadSrVol_cm(radIdxFound_UserInputDia);
```

```
    fprintf(['\tTorroidal Method: Expected flame radius at end: %.3f cm, while',...
        ' last aparent radius detected: %.3f cm\n'], ...
        expectApparentRadiusTorroidBurned_cm, lastApparentRadiusDetected_cm)
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pressure and matching flame radius data Time Shift
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pressureFlameRadiusTimeOffset_ms = 3.9; % Pulse and Radius Time Spark Offset

% Offset time to match 0=ignition time
PSPostProcData.time_scaled_Radius_ms= ...
    PSRunData.timeReal_interp*1000 - PSRunData.timeReal_interp(1)*1000;
    % Subtract off time until ignition so ignition is at t=0, already
    % cut off first bit of radius so just bring back to 0 and scale

PSPostProcData.time_scaled_pressure_RadiusMatched_ms = ...
    PSScopeData.scaledScopeData_ms_all - (pressureFlameRadiusTimeOffset_ms);
    % Subtract off time until ignition so ignition is at t=0

PSPostProcData.idx_pressureSplineRadiusMatchTimeStart = ...
    dsearchn(PSPostProcData.time_scaled_pressure_RadiusMatched_ms, 0);
if PSPostProcData.idx_pressureSplineRadiusMatchTimeStart==1
    fprintf('Error in finding PSPostProcData.idx_pressureSplineRadiusMatchTimeStart\n')
end

PSPostProcData.idx_pressureSplineRadiusMatchTimeEnd = ....
    dsearchn(PSPostProcData.time_scaled_pressure_RadiusMatched_ms, ...
    PSPostProcData.time_scaled_Radius_ms(end));
if PSPostProcData.idx_pressureSplineRadiusMatchTimeEnd == ...
        length(PSPostProcData.time_scaled_pressure_RadiusMatched_ms)
    fprintf(['WARNING in finding PSPostProcData.time_scaled_pressure_RadiusMatched_ms,',...
        ' Pressure captured not long enough...\n'])
end

PSPostProcData.resampledRadiusDataPressureMatch_cm = ...
    spline(PSPostProcData.time_scaled_Radius_ms, PSPostProcData.r_cm_interp, ...
    PSPostProcData.time_scaled_pressure_RadiusMatched_ms(...
        PSPostProcData.idx_pressureSplineRadiusMatchTimeStart:...
        PSPostProcData.idx_pressureSplineRadiusMatchTimeEnd) );
PSPostProcData.resampledRadiusData_cm = ...
    zeros(size(PSPostProcData.time_scaled_pressure_RadiusMatched_ms));
PSPostProcData.resampledRadiusData_cm(PSPostProcData.idx_pressureSplineRadiusMatchTimeStart:...
    PSPostProcData.idx_pressureSplineRadiusMatchTimeEnd) ...
    = PSPostProcData.resampledRadiusDataPressureMatch_cm;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pressure Filtering for overall smooth pressure and Pressure Rise Rate Calcs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if nnz(PSScopeData.pressureScopeUnfiltered_atm_all)>1
    SG_order=3;
    SG_framelen=2501;
    %5001 works really well but is 3.7% of the full length for the entire pressure trace
    % works really well for ~200ms of pressure data
    PSPostProcData.pressureScopeRefiltered_atm_all = ...
        sgolayfilt(PSScopeData.pressureScopeUnfiltered_atm_all, SG_order, SG_framelen);
else
    PSPostProcData.pressureScopeRefiltered_atm_all = ...
        PSScopeData.pressureScopeUnfiltered_atm_all;
end
```

Function: initPSLoopingParam

```
function PSLoopingParam = initPSLoopingParam(PSConditions, PSUsrInput, PSFlags)
% Sets initial loop parameters

%%%% DEBUG Tracking Row
PSLoopingParam.trackingRow = int64(PSUsrInput.pinsCenter_y);
PSLoopingParam.debugStopFrame = 1; %47;
% Conditional Breakpoint: PSLoopingParam.iCurrentFrame >=PSLoopingParam.debugStopFrame && PSRowData.frameCol ==
      PSLoopingParam.trackingRow
%%%% Debug tracking for plotting to make errors easeier to find..

PSLoopingParam.pinsCenter_x =PSUsrInput.pinsCenter_x;
PSLoopingParam.pinsCenter_y =PSUsrInput.pinsCenter_y;
% Number of points to track, updated depending on number of ignighted points
% Number of points to start with for initial ignition, will grow as needed
% Prior = above pin center
PSLoopingParam.numRowPrior = 25;
PSLoopingParam.numRowPriorMAX = 1024;%600;

% Post = below pin center
PSLoopingParam.numRowPost = 25;
PSLoopingParam.numRowPostMAX = 1024; %600;

PSLoopingParam.flameRadDevTol = 20;
    % Number of pixels before radius deviation is flagged, plotting in yellow

if PSFlags.saveFrameStills
    if ~exist(fullfile(strcat(PSConditions.folderPath, filesep,'PostProcessVideo')), 'dir')
        mkdir(fullfile(strcat(PSConditions.folderPath, filesep,'PostProcessVideo')));
    end
end
if PSFlags.saveOrigStills
    if ~exist(fullfile(strcat(PSConditions.folderPath, filesep,'OriginalFrames')), 'dir')
        mkdir(fullfile(strcat(PSConditions.folderPath, filesep,'OriginalFrames')));
    end
end
PSLoopingParam.testVectEndIdx=-1; % Set index, but error if not set later...
if PSFlags.useMatlabVideoReader
    PSLoopingParam.HSCmovObj=VideoReader(fullfile(strcat(PSConditions.folderPath, ...
        filesep,PSConditions.rawVideofilename)));
            % Reread Mov objct as requested from using read...
else
    PSLoopingParam.HSCmovObj=PSVideoReader(fullfile(strcat(PSConditions.folderPath, ...
        filesep,PSConditions.rawVideofilename)));
        % Use Modified VideoReader for faster read
end
PSLoopingParam.iCurrentFrame=1;
PSLoopingParam.frame_orig=0; % Current frame, in original read video format
PSLoopingParam.firstFrame=0; % First frame read for subtraction, etc.
PSLoopingParam.frameVidEnd=...
    PSLoopingParam.HSCmovObj.duration*PSLoopingParam.HSCmovObj.frameRate;
        % End video initial value,
        %modified to PSLoopingParam.frameVidEnd_flameEndAdd above when flame end detected
PSLoopingParam.frameVidEnd_flameEndAdd=10;
    % Amount to add to PSLoopingParam.frameVidEnd when flame edge detected
PSLoopingParam.hitEdge=false; % For PSRunData.dumbflameloc
PSLoopingParam.autoHitEdge=false; % For PSRunData.autoflameloc
    % When hit right most side, end looking for right most black
% Flame jump capping initialization
PSLoopingParam.isFlameStart=false;
    % Flag for when flame initiates from a jump in flame detection!
PSLoopingParam.timesPrevJumpedSoModAdv=0;
    % Number of times artificially modifying flame speed to adjust
    % to allow flame to catch up when previous flame jumped past flame
PSLoopingParam.unPhysicalPixJump= ...
    floor(PSConditions.maxFlameSpeedPhysical_cms*PSUsrInput.pixel2cm/PSConditions.fps);
    % [pix/frame], # of pixels corresponding to PSLoopingParam.maxPixJump_cms
PSLoopingParam.maxPixJump_cms=...
    PSLoopingParam.unPhysicalPixJump*PSConditions.fps/PSUsrInput.pixel2cm;
    %back calculate what will be used...
    %PSLoopingParam.maxPixJump_cms=PSConditions.maxFlameSpeedPhysical_cms; % cm/s!
if ~PSFlags.suppressDebugOutput
    fprintf(['\tSet an Unphysical pixel jump of %.2f, ',...
        'corresponding to a flame propigating at %.2f cm/s\n'],...
        PSLoopingParam.unPhysicalPixJump, PSLoopingParam.maxPixJump_cms)
end
if PSConditions.isBubble
    if PSConditions.phiInit > .9 && PSConditions.phiInit < 1.1
        % if near stoich, need more advancement tolerance
        PSLoopingParam.firstKernelPixFlameJumpTolerance=80;
            % jump tolerance for initial flame kernal development
    else
        PSLoopingParam.firstKernelPixFlameJumpTolerance=70;
            % jump tolerance for initial flame kernal development
    end
else
    if PSConditions.phiInit > .9 && PSConditions.phiInit < 1.1
        % if near stoich, need more advancement tolerance
        PSLoopingParam.firstKernelPixFlameJumpTolerance=100;
            % jump tolerance for initial flame kernal development
    else
        PSLoopingParam.firstKernelPixFlameJumpTolerance=60;
            % jump tolerance for initial flame kernal development
    end
end
PSLoopingParam.initflameKernalDevelFrameNumb=1000;
    % Increased to allow past reset frames to catch their flame
PSLoopingParam.flameKernalDevelFrameNumb=PSLoopingParam.initflameKernalDevelFrameNumb;
    % Set an inital value for detecting initial flame kernal,set to 20 after flame detected
PSLoopingParam.flameKernalDevelFrameAdjust=4;
    % # of frames to allow large jumping to follow initial flame kernal & ignition effects
PSLoopingParam.flameSlipTolerance=1;
    % Pixels/data points, allows for some slip backwards (bubble disturbance) in the flame
    % by this amount of pixels/data points
PSLoopingParam.BubblePopedStartedFrame=0;
PSLoopingParam.stallThresh=.1;
PSLoopingParam.minBrightSlopeRise=.075;
PSLoopingParam.LostFrameTol=1;
    % Number of frames to allow flame to not be tracked before being reset
PSLoopingParam.numRowBubblePopThresh = 10;
PSLoopingParam.prevNumbRowsBubblePopped = 0;
PSLoopingParam.initialCDataSize=0;
```

Function: getPSRadiusVelocity

```
function DrDt = getPSRadiusVelocity(time, radius)
% (timeReal_interp, r_cm_interp_SG)
```

```
DrDt = zeros(length(time)-1, 1);
DrDt(1) = (radius(2)-radius(1))/(time(2)-time(1));

for j = 2:length(radius)-1
    % 5 point stencil
    if j>2 && j<length(radius)-2
        DrDt(j) = (...
            -radius(j+2)+8*radius(j+1)-8*radius(j-1)+radius(j-2)...
        )/...
            ( (time(j+2)-time(j-2))*3 );
                %Assuming average timestepchange, 12*k=(4)*3
    else
        % Else use centered difference
        DrDt(j) = (mean([radius(j+1),radius(j)])-mean([radius(j-1), radius(j)]))/...
            (mean([time(j+1),time(j)])-mean([time(j),time(j-1)])); %drdt_c(i);
    end
end
```

Function: getPSTBurntNSLo

```
function [TBurnt, SLoSimulated, r_cm_SR_rStart, r_cm_SR_rEnd] = getPSTBurntNSLo(passedPhi)
% Output burned temperature for density comp. and simulated planar laminar flame speed

PhiList=[];
TBurntList=[];
SLoSimulatedList=[];

PhiList(end+1)=0.65;
    TBurntList(end+1)=1748.0;
    SLoSimulatedList(end+1)=15.221;

PhiList(end+1)=0.7;
    TBurntList(end+1)=1832.8;
    SLoSimulatedList(end+1)=19.107;

PhiList(end+1)=0.8;
    TBurntList(end+1)=1991.4;
    SLoSimulatedList(end+1)=27.175;

PhiList(end+1)=0.9;
    TBurntList(end+1)=2129.3;
    SLoSimulatedList(end+1)=34.041;

PhiList(end+1)=1.0;
    TBurntList(end+1)=2221.2;
    SLoSimulatedList(end+1)=38.078;

PhiList(end+1)=1.1;
    TBurntList(end+1)=2205.4;
    SLoSimulatedList(end+1)=39.001;

PhiList(end+1)= 1.2;
    TBurntList(end+1)=2153.9;
    SLoSimulatedList(end+1)=34.387;

PhiList(end+1)=1.3;
    TBurntList(end+1)=2073.9;
    SLoSimulatedList(end+1)=24.099;

PhiList(end+1)=1.4;
```

```
    TBurntList(end+1)=1995.3;
    SLoSimulatedList(end+1)=13.672;

PhiList(end+1)=1.5;
    TBurntList(end+1)=1919.1;
    SLoSimulatedList(end+1)=10.286;

PhiExactTolerance=.01;
if abs(passedPhi - 0.65) <= PhiExactTolerance
    % TBurnt=1745.6; % K, Tp from premix, ~adiabaticFlameTemp,
    TBurnt=1748.0; % K, Tp from equilib constant H (Adiabatic) & P(No real pressure change) on Firebrand, ~
        Equilibrium_Burned_Temp
    SLoSimulated=15.221; % cm/s
    r_cm_SR_rStart=1.5;
    r_cm_SR_rEnd=3.5;
elseif abs(passedPhi - 0.7) <= PhiExactTolerance
    % TBurnt=1830.7; % K, Tp from premix, ~adiabaticFlameTemp,
    TBurnt=1832.8; % K, Tp from equilib, ~Equilibrium_Burned_Temp
    SLoSimulated= 19.107; % cm/s
    r_cm_SR_rStart=1.0;
    r_cm_SR_rEnd=3.5;
elseif abs(passedPhi - 0.8) <= PhiExactTolerance
    % TBurnt= 1982.2; % K, Tp from premix, ~adiabaticFlameTemp,
    TBurnt=1991.4; % K, Tp from equilib, ~Equilibrium_Burned_Temp
    SLoSimulated= 27.175; % cm/s
    r_cm_SR_rStart=1.0;
    r_cm_SR_rEnd=3.5;
elseif abs(passedPhi - 0.9) <= PhiExactTolerance
    % TBurnt= 2108.6; % K, Tp from premix, ~adiabaticFlameTemp,
    TBurnt=2129.3; % K, Tp from equilib, ~Equilibrium_Burned_Temp
    SLoSimulated= 34.041; % cm/s
    r_cm_SR_rStart=1.0;
    r_cm_SR_rEnd=3.5;
elseif abs(passedPhi - 1.0) <= PhiExactTolerance
    % TBurnt=2158.3; % K, Tp from premix, ~adiabaticFlameTemp,
    TBurnt=2221.2; % K, Tp from equilib, ~Equilibrium_Burned_Temp
    SLoSimulated=38.078; % cm/s
    r_cm_SR_rStart=2.25;
    r_cm_SR_rEnd=3.5;
elseif abs(passedPhi - 1.1) <= PhiExactTolerance
    % TBurnt= 2169.6; % K, Tp from premix, ~adiabaticFlameTemp,
    TBurnt=2205.4; % K, Tp from equilib, ~Equilibrium_Burned_Temp
    SLoSimulated= 39.001; % cm/s
    r_cm_SR_rStart=1.5;
    r_cm_SR_rEnd=3.5;
elseif abs(passedPhi - 1.2) <= PhiExactTolerance
    % TBurnt= 2113.9; % K, Tp from premix, ~adiabaticFlameTemp,
    TBurnt=2153.9; % K, Tp from equilib, ~Equilibrium_Burned_Temp
    SLoSimulated= 34.387; % cm/s
    r_cm_SR_rStart=1.0;
    r_cm_SR_rEnd=3.5;
elseif abs(passedPhi - 1.3) <= PhiExactTolerance
    % TBurnt= 2056.7; % K, Tp from premix, ~adiabaticFlameTemp,
    TBurnt=2073.9; % K, Tp from equilib, ~Equilibrium_Burned_Temp
    SLoSimulated= 24.099; % cm/s
    r_cm_SR_rStart=1.0;
    r_cm_SR_rEnd=3.5;
elseif abs(passedPhi - 1.4) <= PhiExactTolerance
    % TBurnt=1971.9; % K, Tp from premix, ~adiabaticFlameTemp,
    TBurnt=1995.3; % K, Tp from equilib, ~Equilibrium_Burned_Temp
    SLoSimulated=13.672; % cm/s
    r_cm_SR_rStart=1.0;
    r_cm_SR_rEnd=3.5;
elseif abs(passedPhi - 1.5) <= PhiExactTolerance
    % TBurnt= 1904.5; % K, Tp from premix, ~adiabaticFlameTemp,
    TBurnt=1919.1; % K, Tp from equilib, ~Equilibrium_Burned_Temp
    SLoSimulated= 10.286; % cm/s
    r_cm_SR_rStart=1.0;
    r_cm_SR_rEnd=3.5;
else
    idxPhiNearest=dsearchn(PhiList(:), passedPhi);
    TBurnt=spline(PhiList, TBurntList, passedPhi);
    SLoSimulated=spline(PhiList, SLoSimulatedList, passedPhi);

    fprintf(['\tPhi not exact, setting to splined value %.2f for',...
        ' TBurnt and %.2f for SLo Simulated, and defaulting on r_cm values\n'],...
        TBurnt, SLoSimulated)
    fprintf('\tNearest to %.2f, values of TBurnt(nearest)=%.2f, SLo Simulated(nearest)=%.2f\n', ...
        PhiList(idxPhiNearest), TBurntList(idxPhiNearest), SLoSimulatedList(idxPhiNearest))
    % Default r_cm values...
    r_cm_SR_rStart=1.0;
    r_cm_SR_rEnd=3.5;
end
```

Function: getPSRowLostFrameAction

```
function PSRowData = getPSRowLostFrameAction(PSRowData, PSLoopingParam)
% Function for resetting frame if something bad happened, aka lost flame

if PSRowData.NumFrameLostFlame >= PSLoopingParam.LostFrameTol
    % If previously lost frame, bubble popping already accounted for
    PSRowData.misFire=true;
    PSRowData.NumFrameLostFlame=0;
else
    PSRowData.NumFrameLostFlame = PSRowData.NumFrameLostFlame + 1;
end
```

Function: getPSLoopingParamUpdateRow

```
function PSRowData = getPSLoopingParamUpdateRow(PSRowData, PSRunData, ...
    PSConditions, PSUsrInput, PSLoopingParam)
% Update Row data with Looping Parameter data
% necessary for multiple rows to talk with one-another

%% Grab row and pix location of interest
% Different treatment of Schlieren Darkness not necessary as flipping in PS Main read section)
PSRowData.ipixLocOfInterest = ...
    transpose(PSUsrInput.pinsCenter_x:size(PSLoopingParam.frame, 2));
    % Tranpose to get in column major format
PSRowData.rowValOfInterest=transpose(PSLoopingParam.frame(PSRowData.frameCol, ...
    int64(PSUsrInput.pinsCenter_x:size(PSLoopingParam.frame, 2))));

%% Propigate (auto)hitEdge if lead flame hit edge
if ~PSRowData.autoHitEdge
    PSRowData.hitEdge = PSLoopingParam.hitEdge;
    PSRowData.autoHitEdge = PSLoopingParam.autoHitEdge;
end

%% Determine if flame has already ended
```

```
PSRowData.frameVidEnd = PSLoopingParam.frameVidEnd;

%% Reset this jump in counter for jumped and modified advancement
% Assume no previous jump, switches to 1 if holding flame speed back
PSRowData.isJumpedSoModAdv=0;

if (PSRowData.isLead)
    PSRowData.initUnPhysPixJump= PSLoopingParam.firstKernelPixFlameJumpTolerance;

elseif (PSRunData.leadPSRow.flameKernalDevelFrameNumb > PSLoopingParam.iCurrentFrame - 10)
    % If starting around the time as the lead flame, no modifications really necessary
    if PSConditions.isBubble
        PSRowData.initUnPhysPixJump= max( ...
          PSRunData.leadPSRow.autoflameloc(PSLoopingParam.iCurrentFrame)-...
          PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1)+ ...
          PSLoopingParam.unPhysicalPixJump, PSLoopingParam.firstKernelPixFlameJumpTolerance);
            % Give flame catching a boost if needs it, in flame kernal region with bubble
    else
        PSRowData.initUnPhysPixJump=...
            PSRunData.leadPSRow.autoflameloc(PSLoopingParam.iCurrentFrame)-...
            PSUsrInput.pinsCenter_x + PSLoopingParam.unPhysicalPixJump;
    end

elseif PSRowData.isFlameStart && PSLoopingParam.iCurrentFrame <= ...
        PSRowData.flameKernalDevelFrameNumb
    % Else if already started, no need to modify much
        PSRowData.initUnPhysPixJump= max(abs(PSLoopingParam.firstKernelPixFlameJumpTolerance ...
            - abs(PSRunData.leadPSRow.flameKernalDevelFrameNumb -...
            PSRowData.flameKernalDevelFrameNumb) ), PSLoopingParam.unPhysicalPixJump);
            % Scale down the jump for non-lead flame tracking that has already started
            % ...but not too much as to go below global unphysical pix jump
else
    % Else if trying to start tracking a long time after
    if ~PSConditions.isBubble || (PSConditions.isBubble && ...
            (~PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame-1) || ...
                PSRowData.BubblePopedFrame) || ...
                PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame-1)-...
                PSUsrInput.pinsCenter_x<=30 )
        % If no bubble to limit by, or if bubble hasn't been detected yet in this row
        PSRowData.initUnPhysPixJump = ...
            PSRunData.leadPSRow.autoflameloc(PSLoopingParam.iCurrentFrame)-...
            PSUsrInput.pinsCenter_x + PSLoopingParam.unPhysicalPixJump;
        % Limit by jump by current lead flame and a jump
    else
        % Need to limit by near bubble
        PSRowData.initUnPhysPixJump = min(...
            PSRunData.leadPSRow.autoflameloc(PSLoopingParam.iCurrentFrame)-...
            PSUsrInput.pinsCenter_x + PSLoopingParam.unPhysicalPixJump,...
            PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame-1)-...
            PSUsrInput.pinsCenter_x-5*PSLoopingParam.unPhysicalPixJump);
    end

end
PSRowData.startIdxPt(max(PSLoopingParam.iCurrentFrame-1,1))= ...
    dsearchn(PSRowData.ipixLocOfInterest, ...
    PSRowData.autoflameloc(max(PSLoopingParam.iCurrentFrame-1,1)));
% Attempt to spline only that is necessary. First determine what is physically possible...
if PSRowData.isFlameStart
    % Looking instad of until end of array, look only until PSLoopingParam.unPhysicalPixJump....
    % Find unphysical pixel location, using rowValOfInterest_Smoothed
    if PSLoopingParam.iCurrentFrame>PSRowData.flameKernalDevelFrameNumb
        %allow some slip with initial frames for flame kernal development
        idxUnphysical = find( PSRowData.ipixLocOfInterest(PSRowData.startIdxPt(...
                max(PSLoopingParam.iCurrentFrame-1,1)):end) ...
           >= PSRowData.autoflameloc(max(PSLoopingParam.iCurrentFrame-1,1)) + ...
           PSLoopingParam.unPhysicalPixJump) ...
           + PSRowData.startIdxPt(max(PSLoopingParam.iCurrentFrame-1,1));
    else
        idxUnphysical = find( PSRowData.ipixLocOfInterest(PSRowData.startIdxPt(...
                max(PSLoopingParam.iCurrentFrame-1,1)):end) ...
           >= PSRowData.autoflameloc(max(PSLoopingParam.iCurrentFrame-1,1)) + ...
           PSRowData.initUnPhysPixJump) ...
           + PSRowData.startIdxPt(max(PSLoopingParam.iCurrentFrame-1,1));
            % Additional flame jump not lead flame
    end
else %if i small, before flame is detected...
    % Find unphysical pixel location
    if PSRowData.possiblySparked
        idxUnphysical = find( PSRowData.ipixLocOfInterest(PSRowData.startIdxPt(...
                max(PSLoopingParam.iCurrentFrame-1,1)):end) ...
           >= PSRowData.possiblySparked + PSRowData.initUnPhysPixJump);
           % Unphysical includes what would have been included from possibly sparked
    else
        idxUnphysical = find( PSRowData.ipixLocOfInterest(PSRowData.startIdxPt(...
                max(PSLoopingParam.iCurrentFrame-1,1)):end) ...
           >= PSUsrInput.pinsCenter_x + PSRowData.initUnPhysPixJump);
    end
end
if isempty(idxUnphysical)
   % If close to the end, just use the end as end
   idxUnphysical = length(PSRowData.ipixLocOfInterest);
end

if PSRowData.isFlameStart ...
        && ( length(PSRowData.startIdxPt(PSRowData.startIdxPt>1))) > 10
    % If flame started and has progressed more than 3 frames...
    smoothStart = max(PSRowData.startIdxPt(max(PSLoopingParam.iCurrentFrame-1,1))-1 - ...
        2*PSLoopingParam.unPhysicalPixJump, 1);
        % Max for making sure start doesn't go out of bounds,
        % Min for making sure end doesn't go out of bounds
else
    % Need to check for mis-fire...
    smoothStart = 1;
end
smoothEnd = min(idxUnphysical(1)+2*PSLoopingParam.unPhysicalPixJump, ...
    length(PSRowData.rowValOfInterest) );
    % Min for making sure end doesn't go out of bounds
if ~PSRowData.autoHitEdge % Most time spent here
    % Only need to get computationally intensive smoothed data spline if getting flame
    PSRowData.rowValOfInterest_Smoothed=transpose(spline( ...
        1:length(PSRowData.rowValOfInterest(smoothStart:smoothEnd)), ...
        smooth(double(PSRowData.rowValOfInterest(smoothStart:smoothEnd))), ...
        (1:(PSRowData.rowValOfInterest_SmoothedMulFact*length(...
        PSRowData.rowValOfInterest(smoothStart:smoothEnd)) ))/...
        PSRowData.rowValOfInterest_SmoothedMulFact));
    PSRowData.ipixLocOfInterest_Smoothed=transpose(spline( ...
        1:length(PSRowData.ipixLocOfInterest(smoothStart:smoothEnd)), ...
        double(PSRowData.ipixLocOfInterest(smoothStart:smoothEnd)), ...
        (1:(PSRowData.rowValOfInterest_SmoothedMulFact*length(...
        PSRowData.ipixLocOfInterest(smoothStart:smoothEnd)) ))/...
        PSRowData.rowValOfInterest_SmoothedMulFact));
end
```

Function: getPSRowDumbFlameLoc

```
function PSRowData=getPSRowDumbFlameLoc(PSRowData,PSConditions,PSUsrInput,PSLoopingParam)
% Get original method of flame tracker with darkness thresholding

k_black=find(im2bw(PSLoopingParam.frame(PSRowData.frameCol,:), ...
    PSConditions.flameCutoffThreshold/255)==0);
    %find which pixels are black along mid_y axis for the flame front
if PSUsrInput.isRightDark
    kr_black=find(k_black>PSUsrInput.pinsCenter_x & k_black<PSUsrInput.pixEdgeOfRIO);
    %finds list of black pixels before right edge
else
    kr_black=find(k_black<PSUsrInput.pinsCenter_x & k_black>PSUsrInput.pixEdgeOfRIO);
    %finds list of black pixels before left edge
end
% Select what to do with the right most pixel found
if isempty(kr_black) && ~PSRowData.hitEdge
    % Test for flame existence before hitting right edge (and after for a smooth flame)
        PSRowData.dumbflameloc(PSLoopingParam.iCurrentFrame)=PSUsrInput.pinsCenter_x;
            %if there are no black pixels in between the pins and right wall, assign x at the pins
elseif PSRowData.hitEdge
    % If previously hit right edge...
        PSRowData.dumbflameloc(PSLoopingParam.iCurrentFrame)=PSUsrInput.pixEdgeOfRIO;
else
        PSRowData.dumbflameloc(PSLoopingParam.iCurrentFrame)=k_black(kr_black(end));
        %flame location is the last black pixel before the right edge
        if abs(PSRowData.dumbflameloc(PSLoopingParam.iCurrentFrame) - ...
                PSUsrInput.pixEdgeOfRIO) < 5 %[pixel] Test for flame end
            PSRowData.hitEdge=true;
                % When hit right most side
        end
end
```

Function: getPSFarthestValidIdx

```
function farthestValidIdx = getPSFarthestValidIdx(PSRowData, PSLoopingParam, ...
    rowValIdxToTry, PSFlags)
% Find last value in list that is an increasing term
% exclude the decrease aparent near the end of the frame window

idxFlameIdxAutoSRising=length(rowValIdxToTry)+1;
if length(rowValIdxToTry)>3
    % If at least 4 points, Can user backward diff of seccond order!
    pixJump(1:length(rowValIdxToTry)+1)=-100;
    backDiffSlope=-100; % get loop started
    while (idxFlameIdxAutoSRising>4) && ((backDiffSlope <= PSLoopingParam.minBrightSlopeRise) ...
            || (pixJump(idxFlameIdxAutoSRising)<=2*PSLoopingParam.stallThresh))
        % If sign of 2nd order finite difference positive, ie rising slope we want
        % Used Equation 9 from geometrictools.com/Documentation/FiniteDifferences.pdf
        idxFlameIdxAutoSRising=idxFlameIdxAutoSRising-1;
        backDiffSlope = ...
            (3*PSRowData.rowValOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising))...
            -4*PSRowData.rowValOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising)-1)...
            +PSRowData.rowValOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising)-2))/2;
            % Separated by 1 pixel so h=1
        autoPickedPixelsloc=...
```

```
            PSRowData.ipixLocOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising));
        pixJump(idxFlameIdxAutoSRising)=...
            autoPickedPixelsloc-PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
    end
    if idxFlameIdxAutoSRising==0
        fprintf('Error determining idxFlameIdxAutoSRising!\n')
    end
elseif length(rowValIdxToTry)==2 || length(rowValIdxToTry)==3
    % Else if just 2-3 points...Just do single order backward diff
    pixJump(1:length(rowValIdxToTry)+1)=-100;
    backDiffSlope=-100; % get loop started
    while (idxFlameIdxAutoSRising>=3) ...
            && ( ( backDiffSlope <= PSLoopingParam.minBrightSlopeRise) ...
            || (pixJump(idxFlameIdxAutoSRising)<=2*PSLoopingParam.stallThresh))
        % If sign of 1st order finite difference positive, ie rising slope we want
        idxFlameIdxAutoSRising=idxFlameIdxAutoSRising-1;
        backDiffSlope = ...
            PSRowData.rowValOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising))...
            -PSRowData.rowValOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising)-1);
            % Separated by 1 pixel so h=1

        autoPickedPixelsloc=...
            PSRowData.ipixLocOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising));
        pixJump(idxFlameIdxAutoSRising)=...
            autoPickedPixelsloc-PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
    end
    % Check for case where not a rising slope, ie. ran out of points
    if idxFlameIdxAutoSRising==0
        fprintf('Error determining idxFlameIdxAutoSRising!\n')
    end
elseif isempty(rowValIdxToTry)
    farthestValidIdx=[];
    return
else
    % Else no need to check point, and just outputs the full length...Debug
    idxFlameIdxAutoSRising=length(rowValIdxToTry);
end
if length(rowValIdxToTry)>3 && idxFlameIdxAutoSRising==4
    % If ended prematurely, Just peak minimal pixel jump greather than 0
    idxFlameIdxAutoSRising = find(pixJump>2*PSLoopingParam.stallThresh, 1, 'last');
    if isempty(idxFlameIdxAutoSRising)
        idxFlameIdxAutoSRising=...
            find(pixJump==max(pixJump(and(pixJump~=-100, ...
                pixJump<=PSLoopingParam.unPhysicalPixJump))), 1, 'last');
    end

end

farthestValidIdx=rowValIdxToTry(idxFlameIdxAutoSRising);
```

Function: getPSClosestValidIdx

```
function closestValidIdx = getPSClosestValidIdx(PSRowData, PSLoopingParam, ...
    rowValIdxToTry, PSFlags)
% Find last value in list that is an increasing term
% exclude the decrease aparent near the end of the frame window

flameIdxAutoSWhile_Max=length(rowValIdxToTry);
idxFlameIdxAutoSRising=0;
if length(rowValIdxToTry)>3
```

```
    % Can use forward diff of seccond order
    pixJump(1)=-100;
    forwardDiffSlope=-100; % get loop started
    while (idxFlameIdxAutoSRising<flameIdxAutoSWhile_Max-3) ...
            && ((forwardDiffSlope<=PSLoopingParam.minBrightSlopeRise) ...
            || (pixJump(idxFlameIdxAutoSRising)<=2*PSLoopingParam.stallThresh))
        % While not at end of points that can be tried and not a positive slope (flame is)
        % and pix jump is not greather than the stall threshold.
        % stallThresh*2 as double isn't perfectly accurate
        idxFlameIdxAutoSRising=idxFlameIdxAutoSRising+1;
        if idxFlameIdxAutoSRising==length(rowValIdxToTry)
            fprintf('Error determining idxFlameIdxAutoSRising!\n')
        end
        autoPickedPixelsloc=...
            PSRowData.ipixLocOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising));
        pixJump(idxFlameIdxAutoSRising)=...
            autoPickedPixelsloc-PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
        forwardDiffSlope = ...
            (-3*PSRowData.rowValOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising))...
            +4*PSRowData.rowValOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising)+1)...
            -PSRowData.rowValOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising)+2))/2;
        % Separated by 1 pixel so h=1
        % If sign of 2nd order finite difference positive, ie rising slope we want
    end
elseif length(rowValIdxToTry)>=2
    % Just do single order forward diff
    pixJump(1)=-100;
    forwardDiffSlope=-100; % get loop started
    while (idxFlameIdxAutoSRising<=flameIdxAutoSWhile_Max-2) ...
            && ((forwardDiffSlope<=PSLoopingParam.minBrightSlopeRise) ...
            || (pixJump(idxFlameIdxAutoSRising)<=2*PSLoopingParam.stallThresh))
        idxFlameIdxAutoSRising=idxFlameIdxAutoSRising+1;
        if idxFlameIdxAutoSRising==length(rowValIdxToTry)
            fprintf('Error determining idxFlameIdxAutoSRising!\n')
        end
        autoPickedPixelsloc=...
            PSRowData.ipixLocOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising));
        pixJump(idxFlameIdxAutoSRising)=...
            autoPickedPixelsloc-PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
        forwardDiffSlope =...
            -PSRowData.rowValOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising))...
            +PSRowData.rowValOfInterest_Smoothed(rowValIdxToTry(idxFlameIdxAutoSRising)+1);
        % Separated by 1 pixel so h=1
        % If sign of 1st order finite difference positive, ie rising slope we want
        idxFlameIdxAutoSRising=idxFlameIdxAutoSRising+1;
    end
elseif isempty(rowValIdxToTry)
    closestValidIdx=[];
    return
else
    % Else no need to check point, and just outputs the full length...?
    fprintf('WARNING: Small rowValIdxToTry in getPSClosestValidIdx, using last Value.\n')
    idxFlameIdxAutoSRising=1;
end
if flameIdxAutoSWhile_Max>3 && idxFlameIdxAutoSRising>=flameIdxAutoSWhile_Max-3
    % If ended early, % Just peak minimal pixel jump greather than 0
    idxFlameIdxAutoSRising = find(pixJump>2*PSLoopingParam.stallThresh, 1, 'first');
    if isempty(idxFlameIdxAutoSRising)
        idxFlameIdxAutoSRising=flameIdxAutoSWhile_Max;
    end
```

```
end
closestValidIdx=rowValIdxToTry(idxFlameIdxAutoSRising);
```

Function: gatherPSRunData

```
function PSRunData = gatherPSRunData(PSRunData, PSConditions, PSUsrInput, ...
    PSLoopingParam, PSPlotDispProp, PSFlags)
% Gather relevant raw RunData

%% Flame Radius Selection: Pin Center, Circle Center, or Merge of the two
if PSConditions.isBubble % ~PSFlags.isTrackCirCenter
    isFlameMerge=false;
    % Whether or not to use the combined pinCenter for early and cirCent for later flame
else
    isFlameMerge=true;
end
% Determine which flame detection to use,
% PSRunData.dumbflameloc, PSRunData.autoflameloc_pinCen, PSRunData.autoflameloc_cirCen
if isFlameMerge
    % Find Merge Point
    pickedIdxMerge=...
        find(PSRunData.circleCenterLeadRow-double(int64(PSUsrInput.pinsCenter_y))==0,...
        1, 'last');
    PSRunData.flameloc=zeros(length(PSRunData.autoflameloc_cirCen),1);
    PSRunData.flameloc(1:pickedIdxMerge, 1)=PSRunData.autoflameloc_pinCen(1:pickedIdxMerge);
    PSRunData.flameloc(pickedIdxMerge:length(PSRunData.autoflameloc_cirCen),1 )=...
        PSRunData.autoflameloc_cirCen(pickedIdxMerge:end);
elseif ~PSFlags.isUseTrackedCirCenter
    % If using pin center, otherwise could use: ~PSFlags.isTrackCirCenter
    PSRunData.flameloc=PSRunData.autoflameloc_pinCen;
else
    % else using circle center
    PSRunData.flameloc=PSRunData.autoflameloc_cirCen;
end

%% Determine cropping points, starting flame radius and when flame stops
PSRunData.timeReal_raw=transpose((1:PSConditions.numFrames)/PSConditions.fps);
    % Transpose so in column major format...

% Determine Radius and crop beginning and end before flame starts to propagate
r_raw_pinCen = PSRunData.autoflameloc_pinCen-PSUsrInput.pinsCenter_x;
    %pixel radius measured from center
r_raw_cirCen = PSRunData.autoflameloc_cirCen-PSUsrInput.pinsCenter_x;
    %pixel radius measured from center
r_raw = PSRunData.flameloc-PSUsrInput.pinsCenter_x;
r_raw_StartZeros = find(r_raw_pinCen <= 10*eps);
    %find points that happen before the flame is detected
r_raw_EndZeros = find(abs(r_raw-r_raw(end)) <= 10*eps);
    %find points that don't change after the flame passes

%Grab the first real datapoint
PSRunData.r_raw_lastStartZero=r_raw_StartZeros(end)+1;

%Grab end where radius stagnates when hit wall/dark spot
%PSRunData.r_raw_firstEndZero=r_raw_EndZeros-1;
PSRunData.r_raw_firstEndZero=r_raw_EndZeros(1)-1;
if PSRunData.r_raw_lastStartZero>= PSRunData.r_raw_firstEndZero
    fprintf('Error in detecting first and last start zero with file %s\n',...
        PSConditions.VideoName)
elseif ~PSFlags.suppressDebugOutput
```

```
    % If fails here, try increasing threshold
    fprintf('Spline Factor: %d, grabbing points from index %d (%.2f pix) to %d (%.2f pix)\n', ...
        PSPlotDispProp.splineResolutionFactor, ...
        PSRunData.r_raw_lastStartZero, r_raw_pinCen(PSRunData.r_raw_lastStartZero), ...
        PSRunData.r_raw_firstEndZero, r_raw_pinCen(PSRunData.r_raw_firstEndZero))
end

% Grab where flame stops and first goes backwards in the case of phi too lean
PSRunData.flamelocDiffs=PSRunData.flameloc(2:end)-PSRunData.flameloc(1:end-1);
if PSConditions.phiPost<0.5
    if PSConditions.isBubble
        startSearchLoc_pix = ...
            PSRunData.PSRows(int64(PSUsrInput.pinsCenter_x)).bubbleEndSearchLoc;
        idxStartSearch = ...
            dsearchn(PSRunData.PSRows(int64(PSUsrInput.pinsCenter_y)).autoflameloc, ...
            startSearchLoc_pix);
    else
        idxStartSearch=PSRunData.r_raw_lastStartZero;
    end
    idxNotMoving = find(...
        PSRunData.flamelocDiffs(idxStartSearch:PSRunData.r_raw_firstEndZero)<=10*eps)...
            +PSRunData.r_raw_lastStartZero-1;
    if ~isempty(idxNotMoving) && idxNotMoving(1) > PSRunData.r_raw_firstEndZero
        fprintf(['Found flame stopped moving at frame %d (radius=%.3f cm), and max travel', ...
          ' at pin center: radius=%.3f\n'],idxNotMoving(1),(PSRunData.flameloc(...
              idxNotMoving(1))-PSUsrInput.pinsCenter_x)/PSUsrInput.pixel2cm, ...
          (max(PSRunData.flameloc)-PSUsrInput.pinsCenter_x)/PSUsrInput.pixel2cm)
        PSRunData.flameStopIdx=idxNotMoving(1);
    else
        PSRunData.flameStopIdx=PSRunData.r_raw_firstEndZero;
    end
else
    PSRunData.flameStopIdx=PSRunData.r_raw_firstEndZero;
end
PSRunData.startSpotPastFlameKernal=...
    int64(3*PSPlotDispProp.splineResolutionFactor*...
        (PSLoopingParam.flameKernalDevelFrameNumb-PSRunData.r_raw_lastStartZero));
        %Plotting start spot to crop off inital jump in radius...
% Re-calculate radius based on average pin center in valid region
PSRunData.x_Aparent_avg = ...
    mean(PSRunData.xCirCenter(PSRunData.r_raw_lastStartZero:...
        int64(PSRunData.r_raw_firstEndZero/2)), 'omitnan');
PSRunData.y_Aparent_avg = mean(PSRunData.yCirCenter(PSRunData.r_raw_lastStartZero:...
        int64(PSRunData.r_raw_firstEndZero/2)), 'omitnan');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Sparked Center - To Do: Use to determine Flame center, smooth to get r_raw
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if PSFlags.dispKeyPlots || PSFlags.savePostPlots
    figHandCirCenter=figure('Visible','On');
    if ~PSFlags.dispKeyPlots
        figHandCirCenter.Visible='off';
    end
    xCirCen_Rel_cropped=...
        PSRunData.xCirCenter(PSRunData.r_raw_lastStartZero:PSRunData.r_raw_firstEndZero)...
            -PSUsrInput.pinsCenter_x;
    yCirCen_Rel_cropped=...
        PSUsrInput.pinsCenter_y-PSRunData.yCirCenter(PSRunData.r_raw_lastStartZero:...
        PSRunData.r_raw_firstEndZero);
    % zero values not on screen. Max is 1024x1024
    xCirCen_onScreen=...
        xCirCen_Rel_cropped+-1.*xCirCen_Rel_cropped.*(abs(xCirCen_Rel_cropped) > 1024);
    yCirCen_onScreen=...
        yCirCen_Rel_cropped+-1.*yCirCen_Rel_cropped.*(abs(yCirCen_Rel_cropped) > 1024);

    subplot(2,1,1)
    plot(PSRunData.r_raw_lastStartZero:PSRunData.r_raw_firstEndZero, xCirCen_onScreen, ...
        '.-b', 'DisplayName', 'x-Spk Center')
    hold on;
    plot([PSRunData.r_raw_lastStartZero, PSRunData.r_raw_firstEndZero], ...
        [1,1].*PSRunData.x_Aparent_avg-PSUsrInput.pinsCenter_x, ...
        'k', 'DisplayName', 'Aparent x-Center Weighted Average')
    axis([-inf, inf, -50, 1024])
    title('Circle Center (Relative to Pin Center) vs Frame')
    ylabel('x Cir. Cen., Rel.'); legend('location', 'best'); legend show

    subplot(2,1,2)
    plot(PSRunData.r_raw_lastStartZero:PSRunData.r_raw_firstEndZero, yCirCen_onScreen, ...
        '.-b', 'DisplayName', 'y-Spk Center')
    hold on;
    plot([PSRunData.r_raw_lastStartZero, PSRunData.r_raw_firstEndZero], ...
        [1,1].*PSRunData.y_Aparent_avg-PSUsrInput.pinsCenter_y, ...
        'k', 'DisplayName', 'Aparent y-Center Weighted Average')
    axis([-inf, inf, -1024/4, 1024/4])
    ylabel('y Cir. Cen., Rel.'); xlabel('Frame'); legend('location', 'best'); legend show

    % Save
    filename =['ApparentCenter_vs_Frame_',PSConditions.rawVideofilename,'.eps'];
    outputVideoPath=fullfile(PSConditions.folderPath, filename);
    if PSFlags.savePostPlots
        saveas(figHandCirCenter, outputVideoPath, 'epsc')
    end
    if strcmp(figHandCirCenter.Visible, 'off')
        close(figHandCirCenter)
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Gather Cropped radius data
r_raw_cropped= r_raw(PSRunData.r_raw_lastStartZero:PSRunData.r_raw_firstEndZero);
if nnz(r_raw_cropped)>0
   % need to mark points that went to the pin center because they lost the flame
   r_raw_cropped(r_raw_cropped==0)=nan;
end
r_raw_cropped_Aparent = ...
    PSRunData.leadPSRow(end).StretchRad(PSRunData.r_raw_lastStartZero:...
        PSRunData.r_raw_firstEndZero);
    % Aparent for Stretch Rate Radius Calc
r_raw_cropped_Aparent_avg = ...
    sqrt((PSRunData.autoflameloc_pinCen(PSRunData.r_raw_lastStartZero:...
            PSRunData.r_raw_firstEndZero)-PSRunData.x_Aparent_avg).^2 ...
        + (double(PSRunData.leadPSRow.frameCol)-PSRunData.y_Aparent_avg).^2);
r_raw_cropped_cirCen = ...
    r_raw_cirCen(PSRunData.r_raw_lastStartZero:PSRunData.r_raw_firstEndZero);
r_raw_cropped_pinCen = ...
    r_raw_pinCen(PSRunData.r_raw_lastStartZero:PSRunData.r_raw_firstEndZero);

PSRunData.timeReal_raw_cropped=...
    PSRunData.timeReal_raw(PSRunData.r_raw_lastStartZero:PSRunData.r_raw_firstEndZero);
PSRunData.r_cm_raw_cropped = r_raw_cropped/PSUsrInput.pixel2cm;
PSRunData.r_cm_raw_Aparent_cropped = r_raw_cropped_Aparent/PSUsrInput.pixel2cm;
    % Aparent for Stretch Rate Radius Calc
PSRunData.r_cm_raw_Aparent_cropped_avg = r_raw_cropped_Aparent_avg/PSUsrInput.pixel2cm;
PSRunData.r_raw_cropped_cirCen = r_raw_cropped_cirCen/PSUsrInput.pixel2cm;
PSRunData.r_raw_cropped_pinCen = r_raw_cropped_pinCen/PSUsrInput.pixel2cm;
PSRunData.timeReal_interp=...
    transpose(linspace(PSRunData.timeReal_raw_cropped(1), PSRunData.timeReal_raw_cropped(end), ...
    length(PSRunData.dumbflameloc)*PSPlotDispProp.splineResolutionFactor));
    % Tranpose so in column major format

%% Plot Radius Comparison of Pin Center vs. Aparent Circle Center vs. Merged Version
if PSFlags.dispKeyPlots || PSFlags.savePostPlots
    figHandRCentsvT=figure('Visible','On'); %plots velocity vs time
    if ~PSFlags.dispKeyPlots
        figHandRCentsvT.Visible='off';
    end
    plot(PSRunData.timeReal_raw_cropped.*1E3, PSRunData.r_cm_raw_cropped, ...
        'r', 'DisplayName', 'PSRunData.r_cm_raw_cropped')
    hold on
    plot(PSRunData.timeReal_raw_cropped.*1E3, PSRunData.r_raw_cropped_cirCen, ...
        'c-.', 'DisplayName', 'PSRunData.r_raw_cropped_cirCen')
    plot(PSRunData.timeReal_raw_cropped.*1E3, PSRunData.r_raw_cropped_pinCen, ...
        'g--','DisplayName', 'PSRunData.r_raw_cropped_pinCen')
    title('Radius Comparison - Tracking Lead Flame on Rise')
    legend('location', 'best', 'Interpreter', 'none')
    legend show
    xlabel('Time [ms]')
    ylabel('Radius [cm]')

    filename =['Flame_RadiusPinCirCenters_vs_Time_',PSConditions.rawVideofilename,...
        '.eps']; %Name of file to be saved
    outputVideoPath=fullfile(PSConditions.folderPath, filename);

    if PSFlags.savePostPlots
        saveas(figHandRCentsvT, outputVideoPath, 'epsc')
    end
    if strcmp(figHandRCentsvT.Visible, 'off')
        close(figHandRCentsvT)
    end
end
```

Function: getPSBubbleInitialCirCenter

```
function [xCirCenter, yCirCenter, CirRad] = getPSBubbleInitialCirCenter(PSConditions, PSUsrInput, PSFlags)
% Output Circle center knowing approximate circle radius, etc.
isPlotFrame=false;

%% Grab Fist Frame
HSCmovObj = VideoReader(fullfile(strcat(...
    PSConditions.folderPath, filesep, PSConditions.rawVideofilename)));
if isPlotFrame
    firstFrameFigHand = figure('Visible', 'On');
end

% Read first frame and flip if necessary
if PSUsrInput.isRightDark
    readFirstframe = read(HSCmovObj,1);
else
    readFirstframe = flip(read(HSCmovObj,1), 2);
```

```
end

if isPlotFrame
   set(0,'currentfigure', firstFrameFigHand)
   imshow(readFirstframe)
end

%% Setup fake looping param
PSLoopingParam = initPSLoopingParam(PSConditions, PSUsrInput, PSFlags);
PSLoopingParam.frame = readFirstframe;

%% Fake modify conditions
PSConditions.numFrames=2;
PSLoopingParam.iCurrentFrame=2;

% Using radius to set limits on bubble circumference, use frame to find bubble center
rowBubSearchAboveRange = 2/3*PSUsrInput.rBubble_cm*PSUsrInput.pixel2cm;
rowBubSearchBelowRange = 5/6*PSUsrInput.rBubble_cm*PSUsrInput.pixel2cm;

%% Estimate location of bubble
xCirCenter = PSUsrInput.pinsCenter_x;
yCirCenter = PSUsrInput.pinsCenter_y;
CirRad = PSUsrInput.pixel_bubble_right_avg-PSUsrInput.pinsCenter_x;

if isPlotFrame
   hold on
   plot(xCirCenter, yCirCenter, 'cd')

end

%% Loop through rows
numSkippedRows=0;
for iRow= int64(PSUsrInput.pinsCenter_y - rowBubSearchAboveRange):...
      int64(PSUsrInput.pinsCenter_y + rowBubSearchBelowRange)
   %initialize row
   PSRunData.PSRows(iRow)=initPSRowData(iRow, PSConditions, PSUsrInput, ...
      PSLoopingParam, readFirstframe);

   % Fake an update to initialize the rest of the data...
   PSRunData.PSRows(iRow).autoflameloc(1:PSLoopingParam.iCurrentFrame)= ...
      PSUsrInput.pinsCenter_x;

   PSRunData.leadPSRow = PSRunData.PSRows(iRow);
   PSRunData.PSRows(iRow)=getPSLoopingParamUpdateRow(PSRunData.PSRows(iRow), ...
      PSRunData, PSConditions, PSUsrInput, PSLoopingParam);

   %determine bubble loc
   row=getPSRowBubbleLoc(PSRunData.PSRows(iRow),PSConditions,PSUsrInput,PSLoopingParam);
   x=row.autoBubbleloc(PSLoopingParam.iCurrentFrame);
   y=double(iRow);

   %% Detect if location is reasonable before saving point, ok since selecting
   if isPSIsPtNearCircle(xCirCenter, yCirCenter, CirRad, x, y)
      x_points(iRow-(PSUsrInput.pinsCenter_y - rowBubSearchAboveRange)+1, 1) = x; % row.autoBubbleloc(
          PSLoopingParam.iCurrentFrame);
      y_points(iRow-(PSUsrInput.pinsCenter_y - rowBubSearchAboveRange)+1, 1) = y; % double(iRow);

      if nnz(x_points)>150 %% Get better estimate of circle center
         % could do better here and just update every so often instead of every time
```

```
            [xCirCenter, yCirCenter, CirRad] = getPSBestFitCircle(x_points(x_points~=0), y_points(y_points~=0))
                ;
        end
        if isPlotFrame
            %if saving it, mark it down
            set(0,'currentfigure', firstFrameFigHand)
            hold on
            plot(row.autoBubbleloc(PSLoopingParam.iCurrentFrame), double(iRow), 'b.')
        end
    else
        numSkippedRows=numSkippedRows+1;
        % Mark it as skipped in red
        if isPlotFrame
            set(0,'currentfigure', firstFrameFigHand)
            hold on
            plot(row.autoBubbleloc(PSLoopingParam.iCurrentFrame), double(iRow), 'r.')
        end
    end
end
fprintf('\tSkipped %d rows out of %d\n', numSkippedRows, length(int64(PSUsrInput.pinsCenter_y -
     rowBubSearchAboveRange):int64(PSUsrInput.pinsCenter_y + rowBubSearchBelowRange)));
[xCirCenter, yCirCenter, CirRad] = getPSBestFitCircle(x_points(x_points~=0), y_points(y_points~=0));

if isPlotFrame
    set(0,'currentfigure', firstFrameFigHand); hold on
    plot(xCirCenter, yCirCenter, 'bo')
    pause(3); close(firstFrameFigHand)
end
```

Function: getPSMinNearBubPt

```
function pixLocOfMinNearBub = getPSMinNearBubPt(PSRowData, PSLoopingParam)

surroundingPtThresh = 100/(PSRowData.ipixLocOfInterest(2)-PSRowData.ipixLocOfInterest(1));
    % +/- Number of pix points for serach for local min

idxBubble = dsearchn(PSRowData.ipixLocOfInterest, ...
    PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame));
    % pointOfInterest = PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame);
if isempty(idxBubble) || idxBubble == 1
    fprintf('Error in finding bubble!\n')
end

% Determine possible bubble locations
searchBubMinStart=max(idxBubble-surroundingPtThresh,1);
searchBubMinEnd=min(idxBubble+surroundingPtThresh, length(PSRowData.rowValOfInterest));
[~,maxIdx] = findpeaks(-smooth(double(...
    PSRowData.rowValOfInterest(searchBubMinStart:searchBubMinEnd))) );
if isempty(maxIdx) % Debug
    pixLocOfMinNearBub=PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame);
    return
end
idxToTry = maxIdx+searchBubMinStart-1;
maxThreshedIdx_Full = maxIdx(PSRowData.rowValOfInterest(idxToTry) < PSRowData.initialMean);

if ~exist('maxThreshedIdx_Full', 'var') || isempty(maxThreshedIdx_Full)
    pixLocOfMinNearBub=PSRowData.autoBubbleloc(PSLoopingParam.iCurrentFrame);
else % Nab min
    [~, idxSmallest] = ...
        min(PSRowData.rowValOfInterest(maxThreshedIdx_Full+searchBubMinStart-1));
    pixLocOfMinNearBub=...
        PSRowData.ipixLocOfInterest(maxThreshedIdx_Full(idxSmallest)+searchBubMinStart-1);
end
```

Function: getPSRowBubbleHoldAdvancement

```
function PSRowData = getPSRowBubbleHoldAdvancement(PSRowData, PSRunData, PSConditions, ...
    PSUsrInput, PSLoopingParam, PSFlags, multPrevDiff)
% If flame slipped back, hold advancement (derivative), do only for cases
% where there is a bubble and as bursting casues some issues with the flame tracking...

PSRowData.nearestDetectedIdx=1;
pointsBackAdvancementAve=min(PSLoopingParam.iCurrentFrame-1-2, 12);
% At most 12, but minimum to capture the initial points when flame slips in beginning

prevDiff=mean(...
    PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1-pointsBackAdvancementAve:...
        PSLoopingParam.iCurrentFrame-1)...
    -PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-2-pointsBackAdvancementAve:...
        PSLoopingParam.iCurrentFrame-2));
if prevDiff>=0
    PSRowData.isJumpedSoModAdv=1;
    PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)=...
        PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1)+prevDiff*multPrevDiff;
        %Relaxation to allow flame to "catch up" as prior step must have advanced the flame too much...
else
    PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame)=...
        PSRowData.autoflameloc(PSLoopingParam.iCurrentFrame-1);
end
```

Function: getPSUsrInput

```
function PSUsrInput = getPSUsrInput(PSConditions, PSFlags)
% Get user input, such as pixel scale, sparking center location, end of detection region
% and initial guess of bubble diameter - auto calculated

%location of help dialogue
PSUsrInput.dia_x_pos=0;
PSUsrInput.dia_y_pos=100;
PSUsrInput.dist_cm = 0.9525; %cm, 3/8 in actual diameter of straw

numPixPtAve = 3; % Number of selected pixels to average over for pixel/cm ratio
numBubbleCheckPtAve = 3; % Number of selected pixels to average over for Bubble ratio

% Figure for grabbing user input
if PSFlags.isRunLoop && (~PSFlags.debug_skipUsrInput || PSFlags.isForceUsrInputBubbleDia)
    HSCmovObj = ...
        VideoReader(fullfile(strcat(PSConditions.folderPath, filesep, ...
            PSConditions.rawVideofilename)));
    usrInputFrameFigHand = figure('Visible', 'On');
end

% % Pixel Scale
%loop captures the points from user which will later be used to track flame
if ~PSFlags.isRunLoop
    % Grab from previous run
    % nab fps
    currentUsrIn = PSUsrInput;
    load(PSConditions.specificRunMatlabFN, 'PSUsrInput')
    % PSUsrInput.dist_pixel PSUsrInput.dist_pixel_avg PSUsrInput.pixel2cm
    if isfield(PSUsrInput, 'dist_pixel')
        currentUsrIn.dist_pixel = PSUsrInput.dist_pixel;
    end
    if isfield(PSUsrInput, 'dist_pixel_avg')
        currentUsrIn.dist_pixel_avg = PSUsrInput.dist_pixel_avg;
    end
    if isfield(PSUsrInput, 'pixel2cm')
        currentUsrIn.pixel2cm = PSUsrInput.pixel2cm;
    end
    PSUsrInput = currentUsrIn;
elseif ~PSFlags.debug_skipUsrInput
    % If grabbing user input
    PSUsrInput.dist_pixel = zeros(numPixPtAve,1);
    for j = 1:numPixPtAve
        usrInputframe = read(HSCmovObj,20+5*j); %Show frames 25 and 30
        set(0,'currentfigure', usrInputFrameFigHand)
        imshow(usrInputframe)
        title(PSConditions.VideoName, 'Interpreter', 'none');
        h = helpdlg({'Select Two Points for Pixel Scale:',... %help dialog instructions
            ' Pt 1: x-position of LHS of Straw',...
            ' Pt 2: x-position of RHS of Straw '});
        set(h, 'position', [PSUsrInput.dia_x_pos PSUsrInput.dia_y_pos 300 100]);
        %makes box bigger [xlocation y location width height]

        set(0,'currentfigure', usrInputFrameFigHand);
        % Sometimes another frame gets posted and set as current herebefore this line
        [x_PixelPos, ~] = ginput_CASmodified(2, [3, 4]); %similar to ginput(2)
        PSUsrInput.dist_pixel(j) = ...
            abs(x_PixelPos(length(x_PixelPos))-x_PixelPos(length(x_PixelPos)-1));
        fprintf('\tSelected Two points %.2f, %.2f that are %.2f pixels apart\n', ...
            x_PixelPos(length(x_PixelPos)), ...
            x_PixelPos(length(x_PixelPos)-1), PSUsrInput.dist_pixel(j))
        delete(h) %close help dialog box
        %close(gcf) %imgHndlUsr.Parent)
    end

    PSUsrInput.dist_pixel_avg = mean(PSUsrInput.dist_pixel);
        %averages pixel length between the two straw lengths
    PSUsrInput.pixel2cm = PSUsrInput.dist_pixel_avg/PSUsrInput.dist_cm;
        %converts pixel to cm

    fprintf('\tSelected Mean pixel2cm Ratio: %.2f\n', PSUsrInput.pixel2cm)
elseif PSFlags.isSpecificRunMatlab || PSFlags.isPrevRunMatlab
    % If Skipping user input and if matlab file exists, load previous value
    if ~PSFlags.isPrevRunMatlabStruct && ~PSFlags.isSpecificRunMatlab ...
            && PSFlags.isPrevRunMatlab
        % Convert
        load(PSConditions.prevRunMatlabFN, 'pixel2cm')
        PSUsrInput.pixel2cm = pixel2cm;
        clear pixel2cm
    elseif ~PSFlags.isSpecificRunMatlab && PSFlags.isPrevRunMatlab
        if ~exist('PSUsrInput', 'var') || ~isfield(PSUsrInput, 'pixel2cm')
            % Prevent overwritting if already input
            load(PSConditions.prevRunMatlabFN, 'PSUsrInput');
        end
    elseif PSFlags.isSpecificRunMatlab
        if ~exist('PSUsrInput', 'var') || ~isfield(PSUsrInput, 'pixel2cm')
            % Prevent overwritting if already input
            load(PSConditions.specificRunMatlabFN, 'PSUsrInput');
        end
        % Error check for cases previously ran that started the specificRunMatlab
        % but no longer in the same structured format...
        if ~exist('PSUsrInput', 'var') || ~isfield(PSUsrInput, 'pixel2cm')
            % Extract isBubble from prior conditions if not loaded before
            load(PSConditions.specificRunMatlabFN, 'pixel2cm');
            PSUsrInput.pixel2cm=pixel2cm;
            clear pixel2cm
        end
    else
        fprintf('\tWARNING: No %s or %s file found.\n', ...
                PSConditions.prevRunMatlabFN, PSFlags.isSpecificRunMatlab)
    end
    fprintf('\t Loaded Pixel2cm Ratio: %.2f\n', PSUsrInput.pixel2cm)
else
    PSUsrInput.pixel2cm = 152.68;
    fprintf('\t Using DEFAULT Pixel2cm Ratio: %.2f\n', PSUsrInput.pixel2cm)
end

% % Flame Endpoint (Start/end) Selection
if ~PSFlags.isRunLoop
    % Grab from previous run
    if ~PSFlags.isSpecificRunMatlab && PSFlags.isPrevRunMatlab
        % Convert
        load(PSConditions.prevRunMatlabFN, 'pinsCenter_x', 'pinsCenter_y')
        currentUsrIn = PSUsrInput;
        if exist('pinsCenter_x', 'var')
            currentUsrIn.pinsCenter_x = pinsCenter_x;
            clear pinsCenter_x
        end
        if exist('pinsCenter_y', 'var')
            currentUsrIn.pinsCenter_y = pinsCenter_y;
            clear pinsCenter_y
        end
        PSUsrInput = currentUsrIn;
    elseif PSFlags.isSpecificRunMatlab
        % If running newer version of matlab file, then just nab fps
        currentUsrIn = PSUsrInput;
        load(PSConditions.specificRunMatlabFN, 'PSUsrInput')
        % PSUsrInput.pinsCenter_x, PSUsrInput.pinsCenter_y
        if isfield(PSUsrInput, 'pinsCenter_x')
            currentUsrIn.pinsCenter_x = PSUsrInput.pinsCenter_x;
        end
        if isfield(PSUsrInput, 'pinsCenter_y')
            currentUsrIn.pinsCenter_y = PSUsrInput.pinsCenter_y;
        end
        PSUsrInput = currentUsrIn;
    end
elseif ~PSFlags.debug_skipUsrInput
    % If grabbing user input
    usrInputframe = read(HSCmovObj,PSConditions.fps/100); %Show frame 100 for reference
    %imgHndlUsr=imshow(usrInput)
    set(0,'currentfigure', usrInputFrameFigHand)
    imshow(usrInputframe)
    title(PSConditions.VideoName, 'Interpreter', 'none');
    %set(t, 'Interpreter', 'none')
    h = helpdlg({'Spark Center between pins:',... %help dialog instructions
        ' Pt: Midpoint between the pins'});
    set(h, 'position', [PSUsrInput.dia_x_pos PSUsrInput.dia_y_pos 300 100]);
        %makes box bigger [xlocation y location width height]
    set(0,'currentfigure', usrInputFrameFigHand);
    [x_midBtPins, y_midBtPins] = ginput_CASmodified(1, [1, 2, 3, 4]); %getpts;
    delete(h) % Close help dialog box

    PSUsrInput.pinsCenter_x=x_midBtPins(length(x_midBtPins)); % x midpoint between pins
    PSUsrInput.pinsCenter_y=y_midBtPins(length(x_midBtPins)); % y midpoint between pins
    fprintf('\tSelected pin center at: (%.2f,%.2f)\n', ...
        PSUsrInput.pinsCenter_x, PSUsrInput.pinsCenter_y);

elseif PSFlags.isSpecificRunMatlab || PSFlags.isPrevRunMatlab
    % If Skipping user input and if matlab file exists, load previous value
    if ~PSFlags.isPrevRunMatlabStruct && ~PSFlags.isSpecificRunMatlab ...
            && PSFlags.isPrevRunMatlab
        % Convert
        load(PSConditions.prevRunMatlabFN, 'pinsCenter_x', 'pinsCenter_y')
        PSUsrInput.pinsCenter_x = pinsCenter_x;
        PSUsrInput.pinsCenter_y = pinsCenter_y;
        clear pinsCenter_x pinsCenter_y
    elseif ~PSFlags.isSpecificRunMatlab && PSFlags.isPrevRunMatlab
        if ~exist('PSUsrInput', 'var') || ~isfield(PSUsrInput, 'pinsCenter_x')
            % Prevent overwritting if already input
            load(PSConditions.prevRunMatlabFN, 'PSUsrInput');
        end
    elseif PSFlags.isSpecificRunMatlab
        if ~exist('PSUsrInput', 'var') || ~isfield(PSUsrInput, 'pinsCenter_x')
            % Prevent overwritting if already input
            load(PSConditions.specificRunMatlabFN, 'PSUsrInput');
        end
        % Error check for cases previously ran that started the specificRunMatlab
        % but no longer in the same structured format...
        if ~exist('PSUsrInput', 'var') || ~isfield(PSUsrInput, 'pinsCenter_x')
            % Extract isBubble from prior conditions if not loaded before
            load(PSConditions.specificRunMatlabFN, 'pinsCenter_x', 'pinsCenter_y');
            PSUsrInput.pinsCenter_x=pinsCenter_x;
            PSUsrInput.pinsCenter_y = pinsCenter_y;
            clear pinsCenter_x pinsCenter_y
        end
    else
        fprintf('\tWARNING: No %s or %s file found.\n', ...
                PSConditions.prevRunMatlabFN, PSFlags.isSpecificRunMatlab)
    end
    fprintf('\t Loaded pin center at: (%.2f,%.2f)\n', ...
        PSUsrInput.pinsCenter_x, PSUsrInput.pinsCenter_y);
else
    PSUsrInput.pinsCenter_x = 451.03;%461.52;
    PSUsrInput.pinsCenter_y = 458.53;%440.54;
    fprintf('\t Using DEFAULT pin center at: (%.2f,%.2f)\n', ...
        PSUsrInput.pinsCenter_x, PSUsrInput.pinsCenter_y);
end

% % Right or left edge
if ~PSFlags.isRunLoop
    % Grab from previous run
    if ~PSFlags.isSpecificRunMatlab && PSFlags.isPrevRunMatlab
        % Convert
        load(PSConditions.prevRunMatlabFN, 'pixEdgeOfRIO')
        currentUsrIn = PSUsrInput;
        if exist('pixEdgeOfRIO', 'var')
            currentUsrIn.pixEdgeOfRIO = pixEdgeOfRIO;
            clear pixEdgeOfRIO
        end
        PSUsrInput = currentUsrIn;
    elseif PSFlags.isSpecificRunMatlab
        % If running newer version of matlab file, then just nab fps
        currentUsrIn = PSUsrInput;
        load(PSConditions.specificRunMatlabFN, 'PSUsrInput')
        % PSUsrInput.pixEdgeOfRIO

        if isfield(PSUsrInput, 'pixEdgeOfRIO')
            currentUsrIn.pixEdgeOfRIO = PSUsrInput.pixEdgeOfRIO;
        end
        PSUsrInput = currentUsrIn;
    end
elseif ~PSFlags.debug_skipUsrInput
    % If grabbing user input

    BW = im2bw(usrInputframe, PSConditions.flameCutoffThreshold/255);
    set(0,'currentfigure', usrInputFrameFigHand)
    imgHndlUsr=imshow(BW);
    title(PSConditions.VideoName, 'Interpreter', 'none');

    h = helpdlg({'Select end of Good Flame Region:',... %help dialog instructions
        ' Pt: White pixel BEFORE outside most point ',...
        ' Check threshold if you do not like image'});
    set(h, 'position', [PSUsrInput.dia_x_pos PSUsrInput.dia_y_pos 300 100]);
        %makes box bigger [xlocation y location width height]

    set(0,'currentfigure', usrInputFrameFigHand)
    [x_flameEndpts, ~] = ginput_CASmodified(1, [1, 3, 4]); %getpts;

    PSUsrInput.pixEdgeOfRIO=x_flameEndpts(length(x_flameEndpts));
        %x value which is right before edge of bomb
        %rightedge is to tell where to stop looking for pixels

    % Detect, based on user input whether dark side is on the left or right
    % PSUsrInput.isRightDark: To specify if right or left vertical knife edge, right
    % is dark is default and previous assumption

    delete(h) %close help dialog box
    fprintf('\tSelected Edge of Flame for Dumb Flame tracking: %.2f\n', ...
        PSUsrInput.pixEdgeOfRIO);


    if PSUsrInput.pixEdgeOfRIO < PSUsrInput.pinsCenter_x
        PSUsrInput.isRightDark=false;
    else
        PSUsrInput.isRightDark=true;
    end

elseif PSFlags.isSpecificRunMatlab || PSFlags.isPrevRunMatlab
    % If Skipping user input and if matlab file exists, load previous value
    if ~PSFlags.isPrevRunMatlabStruct && ~PSFlags.isSpecificRunMatlab ...
            && PSFlags.isPrevRunMatlab
        % Convert
        load(PSConditions.prevRunMatlabFN, 'pixEdgeOfRIO') %
        PSUsrInput.pixEdgeOfRIO = pixEdgeOfRIO;
        clear pixEdgeOfRIO
    elseif ~PSFlags.isSpecificRunMatlab && PSFlags.isPrevRunMatlab
        if ~exist('PSUsrInput', 'var') || ~isfield(PSUsrInput, 'pixEdgeOfRIO')
            % Prevent overwritting if already input
            load(PSConditions.prevRunMatlabFN, 'PSUsrInput');
        end
    elseif PSFlags.isSpecificRunMatlab
        if ~exist('PSUsrInput', 'var') || ~isfield(PSUsrInput, 'pixEdgeOfRIO')
            % Prevent overwritting if already input
            load(PSConditions.specificRunMatlabFN, 'PSUsrInput');
        end
        % Error check for cases previously ran that started the specificRunMatlab
        % but no longer in the same structured format...
        if ~exist('PSUsrInput', 'var') || ~isfield(PSUsrInput, 'pixEdgeOfRIO')
            % Extract isBubble from prior conditions if not loaded before
            load(PSConditions.specificRunMatlabFN, 'pixEdgeOfRIO');
            PSUsrInput.pixEdgeOfRIO=pixEdgeOfRIO;
            clear pixEdgeOfRIO
        end
    else
        fprintf('\tWARNING: No %s or %s file found.\n', ...
                PSConditions.prevRunMatlabFN, PSFlags.isSpecificRunMatlab)
    end
    fprintf('\t Loaded Edge of Flame for Dumb Flame tracking: %.2f\n', ...
        PSUsrInput.pixEdgeOfRIO);
else
    PSUsrInput.pixEdgeOfRIO = 983.27;%941.29;%947.29; %870.2793
    fprintf('\t DEFAULT Edge of Flame for Dumb Flame tracking: %.2f\n', ...
        PSUsrInput.pixEdgeOfRIO);
end

if PSUsrInput.pixEdgeOfRIO > 1024 ...
      - 2*floor(PSConditions.maxFlameSpeedPhysical_cms*PSUsrInput.pixel2cm/PSConditions.fps)
    fprintf(['\tWARNING: Given pixel end region of interest is too large',...
        ' and too close to edge of frame, reducing\n'])
    PSUsrInput.pixEdgeOfRIO = PSUsrInput.pixEdgeOfRIO - ...
        floor(PSConditions.maxFlameSpeedPhysical_cms*PSUsrInput.pixel2cm/PSConditions.fps);
end

if PSConditions.isBubble
    if ~PSFlags.isRunLoop
        % Do nothing as going to over-ride this in load call
    elseif ~PSFlags.debug_skipUsrInput || PSFlags.isForceUsrInputBubbleDia
        % If grabbing user input
        PSUsrInput.dist_pixel_bubble = zeros(numBubbleCheckPtAve,1);
        for j = 1:numBubbleCheckPtAve
            usrInputframe = read(HSCmovObj,1+5*j); %Show frames 1 and 6
            set(0,'currentfigure', usrInputFrameFigHand)
            imshow(usrInputframe)
            title(PSConditions.VideoName, 'Interpreter', 'none');
            %set(t, 'Interpreter', 'none')
            h = helpdlg({'Select Two Points for Bubble Diameter:',...
                ' Pt 1: x-position of LHS of Bubble',...
                ' Pt 2: x-position of RHS of Bubble'});
            set(h, 'position', [PSUsrInput.dia_x_pos PSUsrInput.dia_y_pos 300 100]);
                %makes box bigger [xlocation y location width height]

            [x_PixelPos, ~] = ginput_CASmodified(2, [3, 4]); %ginput(2)% getpts;

            PSUsrInput.pixel_bubble_left(j) = x_PixelPos(length(x_PixelPos)-1);
            PSUsrInput.pixel_bubble_right(j) = x_PixelPos(length(x_PixelPos));
            PSUsrInput.d_pixel_bubble(j) = ...
                abs(PSUsrInput.pixel_bubble_left(j)-PSUsrInput.pixel_bubble_right(j));
            fprintf('\tSelected Two points %.2f, %.2f that are %.2f pixels apart\n', ...
                PSUsrInput.pixel_bubble_left(j), PSUsrInput.pixel_bubble_right(j), ...
                PSUsrInput.d_pixel_bubble(j))

            delete(h) %close help dialog box
            %close(gcf) %imgHndlUsr.Parent)
        end

        PSUsrInput.d_pixel_bubble_avg = mean(PSUsrInput.d_pixel_bubble);
            %averages pixel distance of bubble
        PSUsrInput.pixel_bubble_left_avg = mean(PSUsrInput.pixel_bubble_left);
            %averages pixel LHS
        PSUsrInput.pixel_bubble_right_avg = mean(PSUsrInput.pixel_bubble_right);
            %averages pixel LHS
        PSUsrInput.dBubble_selected_cm = PSUsrInput.d_pixel_bubble_avg/PSUsrInput.pixel2cm;
            %converts pixel to cm
        fprintf('\tSelected Bubble Diameter: %.2f cm, left: %.2f, right: %.2f\n', ...
            PSUsrInput.dBubble_selected_cm, PSUsrInput.pixel_bubble_left_avg, ...
            PSUsrInput.pixel_bubble_right_avg);

    elseif PSFlags.isSpecificRunMatlab || PSFlags.isPrevRunMatlab
        % If Skipping user input and if matlab file exists, load previous value
        if ~PSFlags.isPrevRunMatlabStruct && ~PSFlags.isSpecificRunMatlab ...
                && PSFlags.isPrevRunMatlab
        % Convert
            load(PSConditions.prevRunMatlabFN, 'dBubble_selected_cm', ...
                'pixel_bubble_left_avg', 'pixel_bubble_right_avg')
            PSUsrInput.pixel_bubble_left_avg = pixel_bubble_left_avg;
            PSUsrInput.pixel_bubble_right_avg = pixel_bubble_right_avg;
            if exist('dBubble_selected_cm', 'var')
                PSUsrInput.dBubble_selected_cm = dBubble_selected_cm;
            end
            clear pixel_bubble_left_avg pixel_bubble_right_avg dBubble_selected_cm

            if PSConditions.isBubble && ~isfield(PSUsrInput, 'dBubble_selected_cm')
                % If running on previous version and there is a bubble
                load(PSConditions.prevRunMatlabFN, 'pixel_bubble_left_avg', ...
                    'pixel_bubble_right_avg', 'dBubble_selected_cm');
                PSUsrInput.pixel_bubble_left_avg = pixel_bubble_left_avg;
                PSUsrInput.pixel_bubble_right_avg = pixel_bubble_right_avg;
                PSUsrInput.dBubble_selected_cm = dBubble_selected_cm;
                clear pixel_bubble_left_avg pixel_bubble_right_avg dBubble_selected_cm

                if ~isfield(PSUsrInput, 'dBubble_selected_cm') ...
                        && (isfield(PSUsrInput, 'pixel_bubble_left_avg') ...
                        && isfield(PSUsrInput, 'pixel_bubble_right_avg'))
                    PSUsrInput.d_pixel_bubble_avg = ...
                      abs(PSUsrInput.pixel_bubble_left_avg-PSUsrInput.pixel_bubble_right_avg);
                    PSUsrInput.dBubble_selected_cm=...
                        PSUsrInput.d_pixel_bubble_avg/PSUsrInput.pixel2cm;
                end
            end
        elseif ~PSFlags.isSpecificRunMatlab && PSFlags.isPrevRunMatlab
            if ~exist('PSUsrInput', 'var') ...
                    || ~isfield(PSUsrInput, 'pixel_bubble_right_avg')
                % Prevent overwritting if already input
                load(PSConditions.prevRunMatlabFN, 'PSUsrInput')
            end
        elseif PSFlags.isSpecificRunMatlab
            if ~exist('PSUsrInput', 'var') ...
                    || ~isfield(PSUsrInput, 'pixel_bubble_right_avg')
                % Prevent overwritting if already input
                load(PSConditions.specificRunMatlabFN, 'PSUsrInput');
            end

            % Error check for cases previously ran that started the specificRunMatlab
            % but no longer in the same structured format...
            if ~exist('PSUsrInput', 'var') || ~isfield(PSUsrInput, 'pixel_bubble_right_avg')
                % Extract isBubble from prior conditions if not loaded before
                load(PSConditions.specificRunMatlabFN, 'pixel_bubble_right_avg', ...
                    'pixel_bubble_left_avg', 'dBubble_selected_cm');
                PSUsrInput.pixel_bubble_right_avg=pixel_bubble_right_avg;
                PSUsrInput.pixel_bubble_left_avg = pixel_bubble_left_avg;
                PSUsrInput.dBubble_selected_cm = dBubble_selected_cm;
                clear pixel_bubble_left_avg pixel_bubble_right_avg dBubble_selected_cm
            end
        else
            fprintf('\tWARNING: No %s or %s file found.\n', ...
                    PSConditions.prevRunMatlabFN, PSFlags.isSpecificRunMatlab)
        end

    else
        PSUsrInput.dBubble_selected_cm = 2.97;
        PSUsrInput.pixel_bubble_left_avg=188.66;
        PSUsrInput.pixel_bubble_right_avg=644.44;
        fprintf('\t Using DEFAULT Bubble Diameter: %.2f cm, left px: %.2f, right px: %.2f\n',...
            PSUsrInput.dBubble_selected_cm, PSUsrInput.pixel_bubble_left_avg, ...
            PSUsrInput.pixel_bubble_right_avg)
    end
    if ~PSFlags.isRunLoop
        % Do nothing as going to over-ride this in load call
    else
        PSUsrInput.dBubble_cm=PSUsrInput.dBubble_selected_cm;
        % Find location of bubble, ~radius as PSUsrInput.pinsCenter_x is considered
        % '0-point'
        if PSUsrInput.isRightDark
            PSUsrInput.rBubble_cm = ...
                (PSUsrInput.pixel_bubble_right_avg - PSUsrInput.pinsCenter_x)...
                /PSUsrInput.pixel2cm;
        elseif PSUsrInput.pixEdgeOfRIO < PSUsrInput.pinsCenter_x
            % Else left is dark and not yet flipped
            PSUsrInput.rBubble_cm = ...
                (PSUsrInput.pinsCenter_x - PSUsrInput.pixel_bubble_left_avg)...
                /PSUsrInput.pixel2cm;
        elseif PSFlags.isForceUsrInputBubbleDia
            % Update the values not already accounted for
            pixel_bubble_right_avg = 1024/2 + (1024/2 - PSUsrInput.pixel_bubble_left_avg);
                %Shouldn't this be left average and flip to right?
            pixel_bubble_left_avg = 1024/2 + (1024/2 - PSUsrInput.pixel_bubble_right_avg);

            % Now overwrite value, since flipping
            PSUsrInput.pixel_bubble_right_avg=pixel_bubble_right_avg;
            PSUsrInput.pixel_bubble_left_avg=pixel_bubble_left_avg;
            PSUsrInput.rBubble_cm = ...
                (PSUsrInput.pixel_bubble_right_avg-PSUsrInput.pinsCenter_x)...
                /PSUsrInput.pixel2cm;
        else
            % Else left is dark and key parameters flipped already
            PSUsrInput.rBubble_cm = ...
                (PSUsrInput.pixel_bubble_right_avg - PSUsrInput.pinsCenter_x)...
```

```
                /PSUsrInput.pixel2cm;
            if PSUsrInput.rBubble_cm<1
                % If messed up on left/right flip conversion. Fix it!
                pixel_bubble_right_avg = 1024/2 ...
                    + (1024/2 - mean(PSUsrInput.pixel_bubble_left));
                pixel_bubble_left_avg = 1024/2 ...
                    + (1024/2 - mean(PSUsrInput.pixel_bubble_right));

                % Now overwrite value, since flipping
                PSUsrInput.pixel_bubble_right_avg=pixel_bubble_right_avg;
                PSUsrInput.pixel_bubble_left_avg=pixel_bubble_left_avg;

                fprintf('NOTE: Correcting left/right bubble edge and rBubble_cm\n')
                pause(300)
                PSUsrInput.rBubble_cm = ...
                    (PSUsrInput.pixel_bubble_right_avg-PSUsrInput.pinsCenter_x)...
                    /PSUsrInput.pixel2cm;
            end
        end

        %% Determine bubble center
        [PSUsrInput.bubbleCenter_x_pixLoc, PSUsrInput.bubbleCenter_y_pixLoc, ...
            PSUsrInput.bubbleCenter_rad_pixLoc] = ...
            getPSBubbleInitialCirCenter(PSConditions, PSUsrInput, PSFlags);

        PSUsrInput.pixel_bubble_right_auto = ...
            PSUsrInput.bubbleCenter_x_pixLoc + PSUsrInput.bubbleCenter_rad_pixLoc;
        PSUsrInput.rBubble_Auto_cm = PSUsrInput.bubbleCenter_rad_pixLoc/PSUsrInput.pixel2cm;
        PSUsrInput.pinCentRadialBubbleLoc_Auto_cm = ...
            (PSUsrInput.pixel_bubble_right_auto-PSUsrInput.pinsCenter_x)/PSUsrInput.pixel2cm;
        PSUsrInput.dBubble_Auto_cm = 2*PSUsrInput.rBubble_Auto_cm;
    end
else
    % Darkness inspecific values when no bubble (No conv. Necessary)
    PSUsrInput.dBubble_cm=PSConditions.dBubble_assumed_cm;
    PSUsrInput.rBubble_cm=PSUsrInput.dBubble_cm/2;
    PSUsrInput.pixel_bubble_right_avg=...
        PSUsrInput.pinsCenter_x+PSUsrInput.rBubble_cm*PSUsrInput.pixel2cm;
end

if PSFlags.isRunLoop && (~PSFlags.debug_skipUsrInput || PSFlags.isForceUsrInputBubbleDia)
    close(usrInputFrameFigHand)
end
```

Bibliography

[1] Osama Badr and Ghazi Karim. “Flame propagation in stratified methane-air mixtures”. In: *J. Fire Sci.* 2.6 (1984), pp. 415–426. ISSN: 15308049. DOI: 10.1177/073490418400200602.

[2] Saravanan Balusamy, Armelle Cessou, and Bertrand Lecordier. “Laminar propagation of lean premixed flames ignited in stratified mixture”. In: *Combust. Flame* 161.2 (2014), pp. 427–437. ISSN: 00102180. DOI: 10.1016/j.combustflame.2013.08.023.

[3] J. K. Bechtold, C. Cui, and M. Matalon. “The role of radiative losses in self-extinguishing and self-wrinkling flames”. In: *Proc. Combust. Inst.* 30.1 (2005), pp. 177–184. ISSN: 15407489. DOI: 10.1016/j.proci.2004.07.031.

[4] Hans Behrens. “Wasserstoffdiffusion und Flammenstruktur”. In: *Naturwissenschaften* 32.40-43 (1944), pp. 297–299. ISSN: 00281042. DOI: 10.1007/BF01475361.

[5] Johannes Brettschneider. “Berechnung des Luftverhältnisses λ von Luft-Kraftstoff-Gemischen und des Einflusses von Meßfehlern auf λ”. In: *Bosch Technische Berichte* 6 (4 1997), pp. 177–186.

[6] François Charru and E. John Hinch. “’Phase diagram’ of interfacial instabilities in a two-layer Couette flow and mechanism of the long-wave instability”. In: *J. Fluid Mech.* 414 (2000), pp. 195–223. ISSN: 00221120. DOI: 10.1017/S002211200000851X.

[7] Zheng Chen. “Effects of radiation and compression on propagating spherical flames of methane/air mixtures near the lean flammability limit”. In: *Combust. Flame* 157.12 (2010), pp. 2267–2276. ISSN: 00102180. DOI: 10.1016/j.combustflame.2010.07.010.

[8] Zheng Chen. “On the accuracy of laminar flame speeds measured from outwardly propagating spherical flames: Methane/air at normal temperature and pressure”. In: *Combust. Flame* 162.6 (2015), pp. 2442–2453. ISSN: 15562921. DOI: 10.1016/j.combustflame.2015.02.012.

[9] Zheng Chen. “On the extraction of laminar flame speed and Markstein length from outwardly propagating spherical flames”. In: *Combust. Flame* 158.2 (2011), pp. 291–300. ISSN: 0010-2180. DOI: 10.1016/j.combustflame.2010.09.001.

[10] Zheng Chen. “Studies on the Initiation , Propagation, and Extinction of Premixed Flames”. PhD thesis. Princeton University, 2009.

[11] M Cowie and Harry Watts. “Diffusion of Methane and Chloromethanes in Air”. In: *Can. J. Chem.* 49.74 (1971).

[12] A. Cuoci et al. “Extinction of laminar, premixed, counter-flow methane/air flames under unsteady conditions: Effect of H2 addition”. In: *Chem. Eng. Sci.* 93 (2013), pp. 266–276. ISSN: 00092509. DOI: 10.1016/j.ces.2013.02.009.

[13] G. Darrieus. *Propagation d’un front de flame.*

[14] I. V. Dyakov et al. “Measurement of adiabatic burning velocity in methane-oxygen-nitrogen mixtures”. In: *Combust. Sci. Technol.* 172.1 (2010), pp. 81–96. ISSN: 00102202. DOI: 10.1080/00102200108935839.

[15] *Dynamics of Curved Fronts.* San Diego, CA: Academic Press, Inc., 1959. ISBN: 0125503555. DOI: 10.1016/b978-0-08-092523-3.50003-4.

[16] Fokion N Egolfopoulos, P. Cho, and C.K. Law. In: ().

[17] Harvey Einbinder. “The hydrodynamic stability of flame fronts”. In: *J. Chem. Phys.* 21.3 (1953), pp. 480–489. ISSN: 00219606. DOI: 10.1063/1.1698931.

[18] Energy Information Administration. “Household Energy Use in California”. In: (2019).

[19] Shigeo Furuno, Satoshi Iguchi, and Tokuta Inoue. “Lean combustion characteristics of locally stratified charge mixture: Basic studies of in-vessel combustion ignited by laser”. In: *JSAE Rev.* 16.4 (1995), pp. 357–361. ISSN: 03894304. DOI: 10.1016/0389-4304(95)00034-5.

[20] C. Galizzi and D. Escudié. “Experimental analysis of an oblique turbulent flame front propagating in a stratified flow”. In: *Combust. Flame* 145 (2006), pp. 621–634. ISSN: 00102180. DOI: 10.1016/j.combustflame.2010.07.008.

[21] Walter Gander, Gene H. Golub, and Rolf Strebel. “Least-Squares Fitting of Circles and Ellipses”. In: 34 (1994), pp. 558–578.

[22] Hyun Geun and Junseok Kim. “Fluids Two-dimensional Kelvin-Helmholtz instabilities of multi-component fluids”. In: *Eur. J. Mech. B/Fluids* 49 (2015), pp. 77–88. ISSN: 0997-7546. DOI: 10.1016/j.euromechflu.2014.08.001.

[23] George K. Giannakopoulos et al. “Consistent definitions of ”Flame Displacement Speed” and ”Markstein Length” for premixed flame propagation”. In: *Combust. Flame* 162.4 (2015), pp. 1249–1264. ISSN: 15562921. DOI: 10.1016/j.combustflame.2014.10.015.

[24] P. Girard et al. “Flame propagation through unconfined and confined hemispherical stratified gaseous mixtures”. In: *Symp. Combust.* 17.1 (1979), pp. 1247–1255. ISSN: 00820784. DOI: 10.1016/S0082-0784(79)80118-6.

[25] Rama Govindarajan and Kirti Chandra Sahu. “Instabilities in Viscosity-Stratified Flow”. In: *Annu. Rev. Fluid Mech.* 46.1 (2014), pp. 331–353. ISSN: 0066-4189. DOI: 10.1146/annurev-fluid-010313-141351.

[26] Erjiang Hu et al. “Measurements of laminar burning velocities and onset of cellular instabilities of methane-hydrogen-air flames at elevated pressures and temperatures”. In: *Int. J. Hydrogen Energy* 34.13 (2009), pp. 5574–5584. ISSN: 03603199. DOI: 10.1016/j.ijhydene.2009.04.058.

[27] T. Kang and D. C. Kyritsis. “Theoretical investigation of flame propagation through compositionally stratified methane-air mixtures”. In: *Combust. Theory Model.* 13.4 (2009), pp. 705–719. ISSN: 13647830. DOI: 10.1080/13647830903093765.

[28] Taekya Kang and Dimitrios C. Kyritsis. “Methane flame propagation in compositionally stratified gases”. In: *Combust. Sci. Technol.* 177.11 (2005), pp. 2191–2210. ISSN: 00102202. DOI: 10.1080/00102200500240836.

[29] Taekyu Kang and Dimitrios C. Kyritsis. “Departure from quasi-homogeneity during laminar flame propagation in lean, compositionally stratified methane-air mixtures”. In: *Proc. Combust. Inst.* 31 I.1 (2007), pp. 1075–1083. ISSN: 15407489. DOI: 10.1016/j.proci.2006.07.051.

[30] G. A. Karim and H. T. Lam. “Ignition and flame propagation within stratified methane-air mixtures formed by convective diffusion”. In: *Symp. Combust.* 21.1 (1988), pp. 1909–1915. ISSN: 00820784. DOI: 10.1016/S0082-0784(88)80427-2.

[31] G. A. Karim and P. Tsang. “Flame propagation through atmospheres involving concentration gradients formed by mass transfer phenomena”. In: *J. Fluids Eng. Trans. ASME* 97.4 (1975), pp. 615–617. ISSN: 1528901X. DOI: 10.1115/1.3448144.

[32] A P Kelley and C K Law. “Nonlinear effects in the extraction of laminar flame speeds from expanding spherical flames”. In: *Combust. Flame* 156.9 (2009), pp. 1844–1851. ISSN: 0010-2180. DOI: 10.1016/j.combustflame.2009.04.004.

[33] L. Landau. “On the Theory of Slow Combustion”. In: *Acta Physicochim. U.R.S.S.* XIX.1 (1944), pp. 403–411. DOI: 10.1016/b978-0-08-092523-3.50044-7.

[34] Chung K. Law. “Combustion at a crossroads: Status and prospects”. In: *Proc. Combust. Inst.* 31 I.1 (2007), pp. 1–29. ISSN: 15407489. DOI: 10.1016/j.proci.2006.08.124.

[35] William Lowry et al. “Laminar Flame Speed Measurements and Modeling of Pure Alkanes and Alkane Blends at Elevated Pressures”. In: *Proc. ASME Turbo Expo 2010* GT2010 (2010).

[36] G. H. Markstein. “Cell structure of propane flames burning in tubes”. In: *J. Chem. Phys.* 17.4 (1949), pp. 428–429. ISSN: 00219606. DOI: 10.1063/1.1747278.

[37] George H Markstein. “Experimental and Theoretical studies of Flame-front Stability”. In: *J. Aeronaut. Sci.* 18 (1951), pp. 199–209. ISSN: 00223654. DOI: 10.1021/jp9531974.

[38] W. J. Massman. “A review of the molecular diffusivities of H2O, CO2, CH4, CO, O3, SO2, NH3, N2O, NO, and NO2 in air, O2 and N2 near STP”. In: *Atmos. Environ.* 32.6 (1998), pp. 1111–1127. ISSN: 13522310. DOI: 10.1016/S1352-2310(97)00391-9.

[39] Xiangwen Meng et al. "Effects of Direct-Current (DC) Electric Fields on Flame Propagation and Combustion Characteristics of Premixed CH4/O2/N2 Flames". In: *Energy & Fuels* 26.Dc (2012), p. 121018151704002. ISSN: 0887-0624. DOI: `10.1021/ef300972g`.

[40] Akram Mohammad and Khalid A. Juhany. "Laminar burning velocity and flame structure of DME/methane + air mixtures at elevated temperatures". In: *Fuel* 245.February (2019), pp. 105–114. ISSN: 00162361. DOI: `10.1016/j.fuel.2019.02.085`.

[41] A. Pires Da Cruz, A. M. Dean, and J. M. Grenda. "A numerical study of the laminar flame speed of stratified methane/air flames". In: *Proc. Combust. Inst.* 28.2 (2000), pp. 1925–1932. ISSN: 15407489. DOI: `10.1016/S0082-0784(00)80597-4`.

[42] Quantis. "Measuring Fashion 2018: Environmental Impact of the Global Apparel and Footwear Industries Study Full report and methodological considerations". In: (2018), p. 65.

[43] R. J. Kee and F. M. Rupley and J. A. Miller and M. E. Coltrin and J. F. Grcar and E. Meeks and H. K. Moffat and A. E. Lutz and G. Dixon- Lewis and M. D. Smooke and J. Warnatz and G. H. Evans and R. S. Larson and R. E. Mitchell and L. R. Petzold and W. C. Reynolds and M. Caracotsios and W. E. Stewart and P. Glarborg and C. Wang and and O. Adigun. *CHEMKIN Collection.* Vol. Release 3.6. San Diego, CA: Reaction Design Inc., 2000. DOI: `10.1007/978-1-4419-6247-8_9395`.

[44] Youngchul Ra. "Laminar Flame Propagation in a Stratified Charge". PhD thesis. Massachusetts Institute of Technology, 1999.

[45] M.V.S. Ranganadham. "Key world energy statistics". In: *IEA Publ.* (2019), p. 101.

[46] K. J. Richards, P. K. Senecal, and E. Pomraning. "CONVERGE Manual (v2.3)". In: (2019).

[47] K. J. Richards, P. K. Senecal, and E. Pomraning. "CONVERGE (v2.3)". In: (2019).

[48] E. S. Richardson et al. "Effects of equivalence ratio variation on lean, stratified methane-air laminar counterflow flames". In: *Combust. Theory Model.* 14.6 (2010), pp. 775–792. ISSN: 13647830. DOI: `10.1080/13647830.2010.490881`.

[49] C Robinson and D B Smith. "The auto-ignition temeperature of Methane". In: *J. Hazard. Mater. Elsevier Sci. Publ. B.V* 8 (1984), pp. 199–203.

[50] G. Rozenchan et al. "Outward propagation, burning velocities, and chemical effects of methane flames up to 60 ATM". In: *Proc. Combust. Inst.* 29.2 (2002), pp. 1461–1470. ISSN: 15407489. DOI: `10.1016/s1540-7489(02)80179-1`.

[51] David P Schmidt. "Laminar Flame Propagation in Mixtures With Compositional Stratification at Small Length Scales". PhD thesis. University of Illinois at Urbana-Champaign, 2011.

[52] L. Selle, T. Poinsot, and B. Ferret. "Experimental and numerical study of the accuracy of flame-speed measurements for methane/air combustion in a slot burner". In: *Combust. Flame* 158.1 (2011), pp. 146–154. ISSN: 00102180. DOI: `10.1016/j.combustflame.2010.08.003`.

[53] Gary S. Settles. *Schlieren and Shadowgraph Techniques: Visualizing Phenomena in Transparent Media.* Verlag Berlin Heidelberg New York: Springer, 2001.

[54] Xian Shi. "Fundamental Processes in Combustion of Stratified Mixtures". PhD thesis. University of California, Berkeley, 2017.

[55] Gregory P. Smith et al. URL: `http://www.me.berkeley.edu/gri_mech/`.

[56] Toni Tahtouh, Fabien Halter, and Christine Mounaïm-Rousselle. In: ().

[57] UNDP. "Human Development Indices and Indicators. 2018 Statistical Update". In: *United Nations Dev. Program.* 27.4 (2018), p. 123.

[58] U.S. EIA. "Annual Energy Outlook 2018 with projections to 2050". In: *Annu. Energy Outlook 2018 with Proj. to 2050* (2018). ISSN: 1387-1811. DOI: `DOE/EIA-0383(2012) U.S.`. arXiv: `arXiv:1011.1669v3`.

[59] Shuang-Feng Wang et al. "Laminar burning velocities and Markstein lengths of premixed methane/air flames near the lean flammability limit in microgravity". In: *Combust. Flame* 157.4 (2010), pp. 667–675. ISSN: 00102180. DOI: `10.1016/j.combustflame.2010.01.006`.

[60] J. Warnatz, U. Maas, and R.W. Dibble. *Combustion: Physical and Chemical Fundamentals, Moldeling and Simulation, Experiments, Pollutant Formation.* 4th ed. Verlag Berlin Heidelberg: Springer, 2006.

[61] P. A. Witherspoon and D. N. Saraf. "Diffusion of methane, ethane, propane, and n-butane in water from 25 to 43°". In: *J. Phys. Chem.* 69.11 (1965), pp. 3752–3755. ISSN: 00223654. DOI: `10.1021/j100895a017`.

[62] P. Wolanski et al. "Detonation of methane-air mixtures". In: *Eighteenth Symp. Combust.* (1981).

[63] Jiacheng Zhang and John Abraham. "A numerical study of laminar flames propagating in stratified mixtures". In: *Combust. Flame* 163 (2016), pp. 461–471. ISSN: 15562921. DOI: `10.1016/j.combustflame.2015.10.020`.

[64] Ruigang Zhou and Simone Hochgreb. "The behaviour of laminar stratified methane/air flames in counterflow". In: *Combust. Flame* 160.6 (2013), pp. 1070–1082. ISSN: 00102180. DOI: `10.1016/j.combustflame.2013.01.023`.